

Python Tutorial (Codes)

Mustafa GERMEC, PhD



TABLE OF CONTENTS

PYTHON TUTORIAL

1	Introduction to Python	4
2	Strings in Python	15
3	Lists in Python	24
4	Tuples in Python	37
5	Sets in Python	46
6	Dictionaries in Python	55
7	Conditions in Python	64
8	Loops in Python	73
9	Functions in Python	84
10	Exception Handling in Python	98
11	Built-in Functions in Python	108
12	Classes and Objects in Python	143
13	Reading Files in Python	158
14	Writing Files in Python	166
15	String Operators and Functions in Python	176
16	Arrays in Python	190
17	Lambda Functions in Python	200
18	Math Module Functions in Python	206
19	List Comprehension in Python	227
20	Decorators in Python	235
21	Generators in Python	249

To my family...



Python Tutorial

Created by Mustafa Germec, PhD

1. Introduction to Python

First code

In [4]:

```
1 import handcalcs.render
```

In [2]:

```
1 # First python output with 'Print' functions
2 print('Hello World!')
3 print('Hi, Python!')
```

Hello World!

Hi, Python!

Version control

In [10]:

```
1 # Python version check
2 import sys
3 print(sys.version) # version control
4 print(sys.winver) # [Windows only] version number of the Python DLL
5 print(sys.gettrace) # get the global debug tracing function
6 print(sys.argv) # keeps the parameters used while running the program we wrote in a list.
7
```

3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]

3.10

<built-in function gettrace>

```
['c:\\Users\\test\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\ipykernel_launcher.py', '--ip=127.0.0.1', '--stdin=9008', '--control=9006', '--hb=9005', '--Session.signature_scheme="hmac-sha256"', '--Session.key=b"ca6e4e4e-b431-4942-98fd-61b49a098170"', '--shell=9007', '--transport="tcp"', '--iopub=9009', '--f=c:\\Users\\test\\AppData\\Roaming\\jupyter\\runtime\\kernel-17668h2JS6UX2d6li.json']
```

help() function

In [11]:

```
1 # The Python help function is used to display the documentation of modules, functions, classes, keywords, etc.
2 help(sys) # here the module name is 'sys'
```

Help on built-in module sys:

NAME

sys

MODULE REFERENCE

<https://docs.python.org/3.10/library/sys.html> (<https://docs.python.org/3.10/library/sys.html>)

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

DESCRIPTION

This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter.

Dynamic objects:

Comment

In [12]:

```

1 # This is a comment, and to write a comment, '#' symbol is used.
2 print('Hello World!') # This line prints a string.
3
4 # Print 'Hello'
5 print('Hello')
```

Hello World!

Hello

Errors

In [13]:

```

1 # Print string as error message
2 frint('Hello, World!')
```

```

-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13804\1191913539.py in <module>
      1 # Print string as error message
----> 2 frint('Hello, World!')
```

NameError: name 'frint' is not defined

In [14]:

```

1 # Built-in error message
2 print('Hello, World!')
```

```

File "C:\Users\test\AppData\Local\Temp\ipykernel_13804\974508531.py", line 2
  print('Hello, World!')
    ^
```

SyntaxError: unterminated string literal (detected at line 2)

In [15]:

```

1 # Print both string and error to see the running order
2 print('This string is printed')
3 frint('This gives an error message')
4 print('This string will not be printed')
```

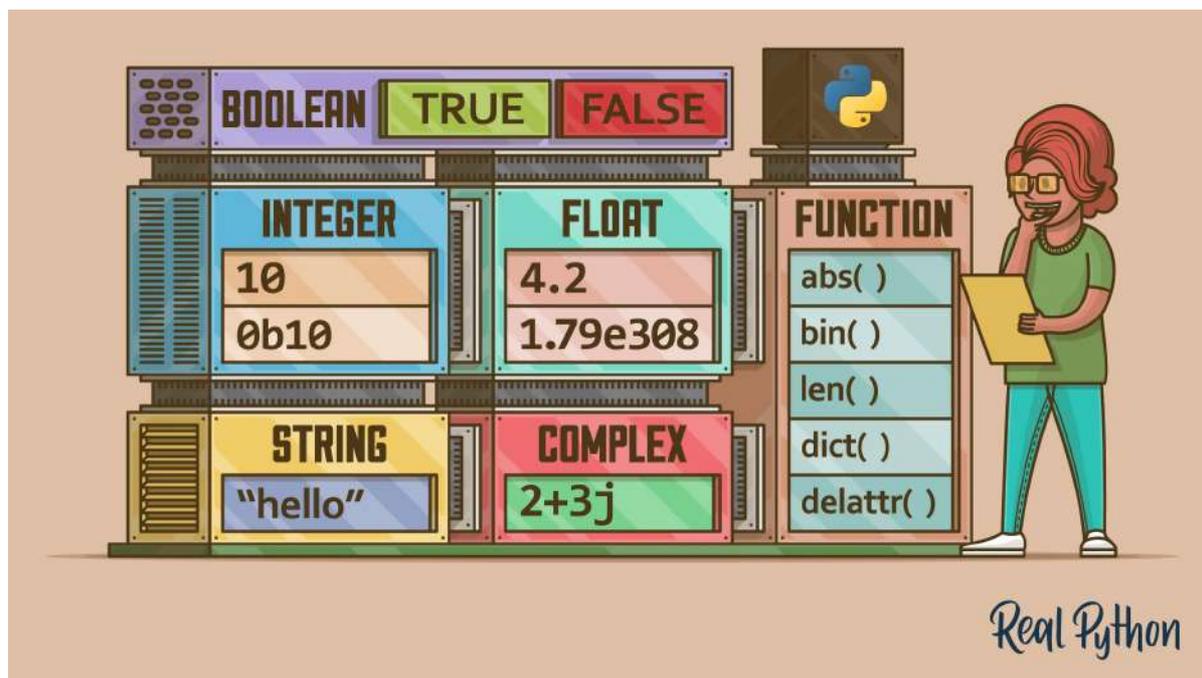
This string is printed

```

-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13804\3194197137.py in <module>
      1 # Print both string and error to see the running order
      2 print('This string is printed')
----> 3 frint('This gives an error message')
      4 print('This string will not be printed')
```

NameError: name 'frint' is not defined

Basic data types in Python



In [27]:

```

1 # String
2 print("Hello, World!")
3 # Integer
4 print(12)
5 # Float
6 print(3.14)
7 # Boolean
8 print(True)
9 print(False)
10 print(bool(1)) # Output = True
11 print(bool(0)) # Output = False
12

```

Hello, World!

12

3.14

True

False

True

False

type() function

In [29]:

```
1 # String
2 print(type('Hello, World!'))
3
4 # Integer
5 print(type(15))
6 print(type(-24))
7 print(type(0))
8 print(type(1))
9
10 # Float
11 print(type(3.14))
12 print(type(0.5))
13 print(type(1.0))
14 print(type(-5.0))
15
16 # Boolean
17 print(type(True))
18 print(type(False))
```

```
<class 'str'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'int'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'float'>
<class 'bool'>
<class 'bool'>
```

In [25]:

```
1 # to obtain the information about 'interger' and 'float'
2 print(sys.int_info)
3 print() # to add a space between two outputs, use 'print()' function
4 print(sys.float_info)
```

```
sys.int_info(bits_per_digit=30, sizeof_digit=4)
```

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585
072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-
16, radix=2, rounds=1)
```

Converting an object type to another object type

In [35]:

```
1 # Let's convert the integer number 6 to a string and a float
2
3 number = 6
4
5 print(str(number))
6 print(float(number))
7 print(type(number))
8 print(type(str(number)))
9 print(type(float(number)))
10 str(number)
```

```
6
6.0
<class 'int'>
<class 'str'>
<class 'float'>
```

Out[35]:

```
'6'
```

In [37]:

```
1 # Let's convert the float number 3.14 to a string and an integer
2
3 number = 3.14
4
5 print(str(number))
6 print(int(number))
7 print(type(number))
8 print(type(str(number)))
9 print(type(int(number)))
10 str(number)
```

```
3.14
3
<class 'float'>
<class 'str'>
<class 'int'>
```

Out[37]:

```
'3.14'
```

In [42]:

```
1 # Let's convert the booleans to an integer, a float, and a string
2
3 bool_1 = True
4 bool_2 = False
5
6 print(int(bool_1))
7 print(int(bool_2))
8 print(float(bool_1))
9 print(float(bool_2))
10 print(str(bool_1))
11 print(str(bool_2))
12 print(bool(1))
13 print(bool(0))
```

```
1
0
1.0
0.0
True
False
True
False
```

In [46]:

```
1 # Let's find the data types of 9/3 and 9//4
2
3 print(9/3)
4 print(9//4)
5 print(type(9/3))
6 print(type(9//4))
```

```
3.0
2
<class 'float'>
<class 'int'>
```

Expersion and variables

In [47]:

```
1 # Addition
2
3 x = 56+65+89+45+78.5+98.2
4 print(x)
5 print(type(x))
```

```
431.7
<class 'float'>
```

In [48]:

```
1 # Substraction
2
3 x = 85-52-21-8
4 print(x)
5 print(type(x))
```

```
4
<class 'int'>
```

In [49]:

```
1 # Multiplication
2
3 x = 8*74
4 print(x)
5 print(type(x))
```

```
592
<class 'int'>
```

In [50]:

```
1 # Division
2
3 x = 125/24
4 print(x)
5 print(type(x))
```

```
5.2083333333333333
<class 'float'>
```

In [51]:

```
1 # Floor division
2
3 x = 125//24
4 print(x)
5 print(type(x))
```

```
5
<class 'int'>
```

In [52]:

```
1 # Modulus
2
3 x = 125%24
4 print(x)
5 print(type(x))
```

```
5
<class 'int'>
```

In [54]:

```
1 # Exponentiation
2
3 x = 2**3
4 print(x)
5 print(type(x))
```

```
8
<class 'int'>
```

In [56]:

```
1 # An example: Let's calculate how many minutes there are in 20 hours?
2
3 one_hour = 60 # 60 minutes
4 hour = 20
5 minutes = one_hour * hour
6 print(minutes)
7 print(type(minutes))
8
9 # An example: Let's calculate how many hours there are in 348 minutes?
10
11 minutes = 348
12 one_hour = 60
13 hours = 348/60
14 print(hours)
15 print(type(hours))
```

```
1200
<class 'int'>
5.8
<class 'float'>
```


In [62]:

```
1 x = 25
2 y = 87
3 z = 5*x - 2*y
4 print(z)
5
6 t = z/7
7 print(t)
8
9 z = z/14
10 print(z)
```

-49

-7.0

-3.5

In [68]:

```
1 x, y, z = 8, 4, 2  # the values of x, y, and z can be written in one line.
2 print(x, y, z)
3 print(x)
4 print(y)
5 print(z)
6 print(x/y)
7 print(x/z)
8 print(y/z)
9 print(x+y+z)
10 print(x*y*z)
11 print(x-y-z)
12 print(x/y/z)
13 print(x//y//z)
14 print(x%y%z)
15
```

8 4 2

8

4

2

2.0

4.0

2.0

14

64

2

1.0

1

0

Python Tutorial

Created by Mustafa Germec, PhD

2. Strings



In [1]:

```
1 # Employ double quotation marks for describing a string
2 "Hello World!"
```

Out[1]:

'Hello World!'

In [2]:

```
1 # Employ single quotation marks for describing a string
2 'Hello World!'
```

Out[2]:

'Hello World!'

In [3]:

```
1 # Digitals and spaces in a string
2 '3 6 9 2 6 8'
```

Out[3]:

'3 6 9 2 6 8'

In [4]:

```
1 # Specific characters in a string
2 '@#5_]*$%^&'
```

Out[4]:

```
'@#5_]*$%^&'
```

In [5]:

```
1 # printing a string
2 print('Hello World!')
```

Hello World!

In [6]:

```
1 # Assigning a string to a variable 'message'
2 message = 'Hello World!'
3 print(message)
4 message
```

Hello World!

Out[6]:

```
'Hello World!'
```

Indexing of a string

In [7]:

```
1 # printing the first element in a string
2
3 message = 'Hello World!'
4 print(message[0])
```

H

In [8]:

```
1 # Printing the element on index 8 in a string
2 print(message[8])
```

r

In [9]:

```
1 # lenght of a string includign spaces
2
3 len(message)
```

Out[9]:

12

In [10]:

```
1 # Printing the last element in a string
2 print(message[11])
3
4 # Another comment writing type is as follows using triple quotes.
5
6 """
7 Although the length of the string is 12, since the indexing in Python starts with 0,
8 the number of the last element is therefore 11.
9 """
```

!

Out[10]:

```
'\nAlthough the length of the string is 12, since the indexing in Python starts with 0, \nthe number of the last element is therefore 11.\n'
```

Negative indexing of a string

In [11]:

```
1 # printing the last element of a string
2
3 message[-1]
```

Out[11]:

'!'

In [12]:

```
1 # printing the first element of a string
2
3 message[-12]
4
5 """
6 Since the negative indexing starts with -1, in this case, the negative index number
7 of the first element is equal to -12.
8 """
```

Out[12]:

```
'\nSince the negative indexing starts with -1, in this case, the negative index number \nof the first element is equal to -12.\n'
```

In [13]:

```
1 print(len(message))
2 len(message)
```

12

Out[13]:

12

In [14]:

```
1 len('Hello World!')
```

Out[14]:

12

Slicing of a string

In [15]:

```
1 # Slicing on the variable 'message' with only index 0 to index 5  
2 message[0:5]
```

Out[15]:

'Hello'

In [16]:

```
1 # Slicing on the variable 'message' with only index 6 to index 12  
2 message[6:12]
```

Out[16]:

'World!'

Striding in a string

In [17]:

```
1 # to select every second element in the variable 'message'  
2  
3 message[::2]
```

Out[17]:

'HloWrD'

In [18]:

```
1 # corporation of slicing and striding  
2 # get every second element in range from index 0 to index 6  
3  
4 message[0:6:2]
```

Out[18]:

'Hlo'

Concatenate of strings

In [19]:

```
1 message = 'Hello World!'
2 question = ' How many people are living on the earth?'
3 statement = message+question
4 statement
```

Out[19]:

'Hello World! How many people are living on the earth?'

In [20]:

```
1 # printing a string for 4 times
2 4*" Hello World!"
```

Out[20]:

' Hello World! Hello World! Hello World! Hello World!'

Escape sequences

In [21]:

```
1 # New line escape sequence
2 print('Hello World! \nHow many people are living on the earth?')
```

Hello World!

How many people are living on the earth?

In [22]:

```
1 # Tab escape sequence
2 print('Hello World! \tHow many people are living on the earth?')
```

Hello World! How many people are living on the earth?

In [23]:

```
1 # back slash in a string
2 print('Hello World! \\ How many people are living on the earth?')
3
4 # r will say python that a string will be show as a raw string
5 print(r'Hello World! \ How many people are living on the earth?')
```

Hello World! \ How many people are living on the earth?

Hello World! \ How many people are living on the earth?

String operations

In [24]:

```
1 message = 'hello python!'
2 print('Before uppercase: ', message)
3
4 # convert uppercase the elements in a string
5 message_upper = message.upper()
6 print('After uppercase: ', message_upper)
7
8 # convert lowercase the elements in a string
9 message_lower = message.lower()
10 print('Again lowercase: ', message_lower)
11
12 # convert first letter of string to uppercase
13 message_title = message.title()
14 print('The first element of the string is uppercase: ', message_title)
```

Before uppercase: hello python!
After uppercase: HELLO PYTHON!
Again lowercase: hello python!
The first element of the string is uppercase: Hello Python!

In [25]:

```
1 # replace() method in a string
2 message = 'Hello Python!'
3 message_hi = message.replace('Hello', 'Hi')
4 message_python = message.replace('Python', 'World')
5 print(message_hi)
6 print(message_python)
```

Hi Python!
Hello World!

In [26]:

```
1 # find() method application in a string
2 message = 'Hello World!'
3 print(message.find('Wo'))
4
5 # the output is the index number of the first element of the substring
```

6

In [27]:

```
1 # find() method application to obtain a substring in a string
2 message.find('World!')
```

Out[27]:

6

In [28]:

```
1 # if cannot find the substring in a string, the output is -1.  
2 message.find('cndsjud')
```

Out[28]:

-1

In [30]:

```
1 text = 'Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had  
2  
3 # find the first index of the substring 'Nancy'  
4 text.find('Nancy')
```

Out[30]:

122

In [31]:

```
1 # replace the substring 'Nancy' with 'Nancy Lier Cosgrove Mullis'  
2 text.replace('Nancy', 'Nancy Lier Cosgrove Mullis')
```

Out[31]:

'Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy Lier Cosgrove Mullis, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.'

In [32]:

```
1 # convert the text to lower case  
2 text.lower()
```

Out[32]:

'jean-paul sartre somewhere observed that we each of us make our own hell out of the people around us. had jean-paul known nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. she will be his morning and his evening star, shining with the brightest and the softest light in his heaven. she will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. their faith in one another will be deeper than time and their eternal spirit will be seamless once again.'

In [33]:

```
1 # convert the first letter of the text to capital letter
2 text.capitalize()
```

Out[33]:

'Jean-paul sartre somewhere observed that we each of us make our own hell out of the people around us. had jean-paul known nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. she will be his morning and his evening star, shining with the brightest and the softest light in his heaven. she will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. their faith in one another will be deeper than time and their eternal spirit will be seamless once again.'

In [34]:

```
1 # casefold() method returns a string where all the characters are in lower case
2 text.casefold()
```

Out[34]:

'jean-paul sartre somewhere observed that we each of us make our own hell out of the people around us. had jean-paul known nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. she will be his morning and his evening star, shining with the brightest and the softest light in his heaven. she will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. their faith in one another will be deeper than time and their eternal spirit will be seamless once again.'

In [35]:

```
1 # center() method will center align the string, using a specified character (space is the default) as the fill character.
2 message = 'Hallo Leute!'
3 message.center(50, '-')
```

Out[35]:

```
'-----Hallo Leute!-----'
```

In [36]:

```
1 # count() method returns the number of elements with the specified value
2 text.count('and')
```

Out[36]:

7

In [37]:

```
1 # format() method
2 """
3 The format() method formats the specified value(s) and insert them inside the string's placeholder.
4 The placeholder is defined using curly brackets: {}.
5 """
6
7 txt = "Hello {word}"
8 print(txt.format(word = 'World!'))
9
10 message1 = 'Hi, My name is {} and I am {} years old.'
11 print(message1.format('Bob', 36))
12
13 message2 = 'Hi, My name is {name} and I am {number} years old.'
14 print(message2.format(name = 'Bob', number = 36))
15
16 message3 = 'Hi, My name is {0} and I am {1} years old.'
17 print(message3.format('Bob', 36))
```

Hello World!

Hi, My name is Bob and I am 36 years old.

Hi, My name is Bob and I am 36 years old.

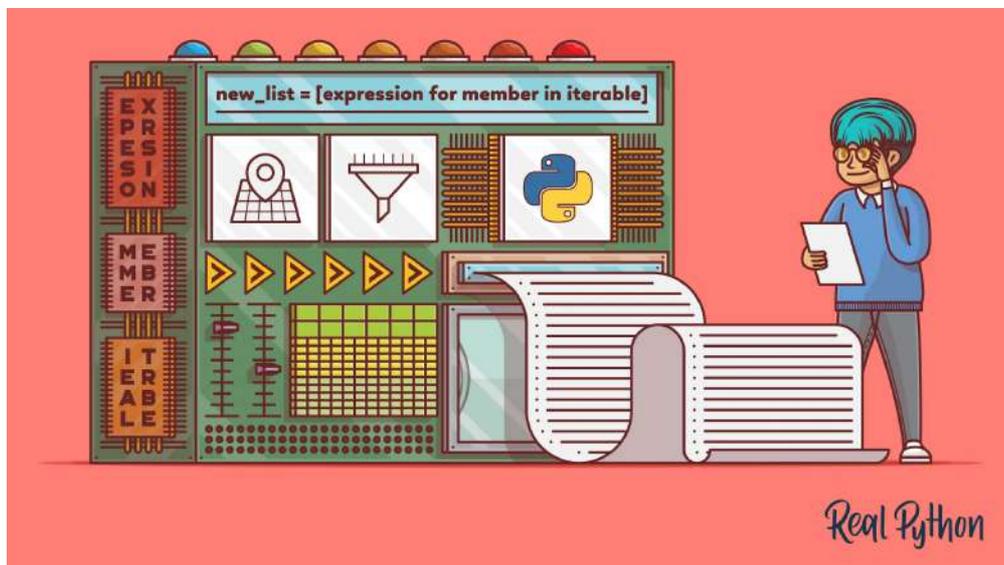
Hi, My name is Bob and I am 36 years old.

Python Tutorial

Created by Mustafa Germec, PhD

3. Lists

- Lists are ordered.
- Lists can contain any arbitrary objects.
- List elements can be accessed by index.
- Lists can be nested to arbitrary depth.
- Lists are mutable.
- Lists are dynamic.



Indexing

In [1]:

```
1 # creatinng a list
2 nlis = ['python', 25, 2022]
3 nlis
```

Out[1]:

```
['python', 25, 2022]
```

In [7]:

```
1 print('Positive and negative indexing of the first element: \n - Positive index:', nlis[0], '\n - Negative index:', nlis[-3])
2 print()
3 print('Positive and negative indexing of the second element: \n - Positive index:', nlis[1], '\n - Negative index:', nlis[-2])
4 print()
5 print('Positive and negative indexing of the third element: \n - Positive index:', nlis[2], '\n - Negative index:', nlis[-1])
```

Positive and negative indexing of the first element:

- Positive index: python
- Negative index: python

Positive and negative indexing of the second element:

- Positive index: 25
- Negative index: 25

Positive and negative indexing of the third element:

- Positive index: 2022
- Negative index: 2022

What can content a list?

- Strings
- Floats
- Integer
- Boolean
- Nested List
- Nested Tuple
- Other data structures

In [8]:

```
1 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
2 nlis
```

Out[8]:

```
['python',
 3.14,
 2022,
 [1, 1, 2, 3, 5, 8, 13, 21, 34],
 ('hello', 'python', 3, 14, 2022)]
```

List operations

In [10]:

```
1 # take a list
2 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
3 nlis
```

Out[10]:

```
['python',
 3.14,
 2022,
 [1, 1, 2, 3, 5, 8, 13, 21, 34],
 ('hello', 'python', 3, 14, 2022)]
```

In [11]:

```
1 # length of the list
2 len(nlis)
```

Out[11]:

5

Slicing

In [20]:

```
1 # slicing of a list
2 print(nlis[0:2])
3 print(nlis[2:4])
4 print(nlis[4:6])
```

```
['python', 3.14]
[2022, [1, 1, 2, 3, 5, 8, 13, 21, 34]]
[('hello', 'python', 3, 14, 2022)]
```

Extending the list

- we use the **extend()** function to add a new element to the list.
- With this function, we add more than one element to the list.

In [25]:

```
1 # take a list
2 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
3 nlis.extend(['hello world!', 1.618])
4 nlis
```

Out[25]:

```
['python',
 3.14,
 2022,
 [1, 1, 2, 3, 5, 8, 13, 21, 34],
 ('hello', 'python', 3, 14, 2022),
 'hello world!',
 1.618]
```

append() method

- As different from the **extend()** method, with the **append()** method, we add only one element to the list
- You can see the difference by comparing the above and below codes.

In [27]:

```
1 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
2 nlis.append(['hello world!', 1.618])
3 nlis
```

Out[27]:

```
['python',
 3.14,
 2022,
 [1, 1, 2, 3, 5, 8, 13, 21, 34],
 ('hello', 'python', 3, 14, 2022),
 ['hello world!', 1.618]]
```

len(), append(), count(), index(), insert(), max(), min(), sum() functions

In [99]:

```
1 lis = [1,2,3,4,5,6,7]
2 print(len(lis))
3 lis.append(4)
4 print(lis)
5 print(lis.count(4))    # How many 4 are on the list 'lis'?
6 print(lis.index(2))   # What is the index of the number 2 in the list 'lis'?
7 lis.insert(8, 9)      # Add number 9 to the index 8.
8 print(lis)
9 print(max(lis))       # What is the maximum number in the list?
10 print(min(lis))      # What is the minimum number in the list?
11 print(sum(lis))      # What is the sum of the numbers in the list?
```

```
7
[1, 2, 3, 4, 5, 6, 7, 4]
2
1
[1, 2, 3, 4, 5, 6, 7, 4, 9]
9
1
41
```

Changing the element of a list since it is mutable

In [31]:

```

1 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
2 print('Before changing:', nlis)
3 nlis[0] = 'hello python!'
4 print('After changing:', nlis)
5 nlis[1] = 1.618
6 print('After changing:', nlis)
7 nlis[2] = [3.14, 2022]
8 print('After changing:', nlis)

```

Before changing: ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]
 After changing: ['hello python!', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]
 After changing: ['hello python!', 1.618, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]
 After changing: ['hello python!', 1.618, [3.14, 2022], [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]

Deleting the element from the list using del() function

In [34]:

```

1 print('Before changing:', nlis)
2 del(nlis[0])
3 print('After changing:', nlis)
4 del(nlis[-1])
5 print('After changing:', nlis)

```

Before changing: [1.618, [3.14, 2022], [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]
 After changing: [[3.14, 2022], [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]
 After changing: [[3.14, 2022], [1, 1, 2, 3, 5, 8, 13, 21, 34]]

In [81]:

```

1 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
2 print('Before deleting:', nlis)
3 del nlis
4 print('After deleting:', nlis)

```

Before deleting: ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]

```

-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13488\2190443850.py in <module>
      2 print('Before deleting:', nlis)
      3 del nlis
----> 4 print('After deleting:', nlis)

```

NameError: name 'nlis' is not defined

Conversion of a string into a list using split() function

In [36]:

```
1 message = 'Python is a programming language.'  
2 message.split()
```

Out[36]:

```
['Python', 'is', 'a', 'programming', 'language.']
```

Use of split() function with a delimiter

In [57]:

```
1 text = 'p,y,t,h,o,n'  
2 text.split(",")
```

Out[57]:

```
['p', 'y', 't', 'h', 'o', 'n']
```

Basic operations

In [90]:

```
1 nlis_1 = ['a', 'b', 'hello', 'Python']  
2 nlis_2 = [1,2,3,4, 5, 6]  
3 print(len(nlis_1))  
4 print(len(nlis_2))  
5 print(nlis_1+nlis_2)  
6 print(nlis_1*3)  
7 print(nlis_2*3)  
8 for i in nlis_1:  
9     print(i)  
10 for i in nlis_2:  
11     print(i)  
12 print(4 in nlis_1)  
13 print(4 in nlis_2)
```

```
4  
6  
['a', 'b', 'hello', 'Python', 1, 2, 3, 4, 5, 6]  
['a', 'b', 'hello', 'Python', 'a', 'b', 'hello', 'Python', 'a', 'b', 'hello', 'Python']  
[1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]  
a  
b  
hello  
Python  
1  
2  
3  
4  
5  
6  
False  
True
```

Copy the list

In [62]:

```
1 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
2 copy_list = nlis
3 print('nlis:', nlis)
4 print('copy_list:', copy_list)
```

nlis: ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]

copy_list: ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]

In [70]:

```
1 # The element in the copied list also changes when the element in the original list was changed.
2 # See the following example
3
4 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
5 print(nlis)
6 copy_list = nlis
7 print(copy_list)
8 print('copy_list[0]:', copy_list[0])
9 nlis[0] = 'hello python!'
10 print('copy_list[0]:', copy_list[0])
```

['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]

['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]

copy_list[0]: python

copy_list[0]: hello python!

Clone the list

In [72]:

```
1 # The cloned list is a new copy or clone of the original list.
2 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
3 clone_lis = nlis[:]
4 clone_lis
```

Out[72]:

```
['python',
 3.14,
 2022,
 [1, 1, 2, 3, 5, 8, 13, 21, 34],
 ('hello', 'python', 3, 14, 2022)]
```

In [74]:

```

1 # When an element in the original list is changed, the element in the cloned list does not change.
2 nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
3 print(nlis)
4 clone_list = nlis[:]
5 print(clone_list)
6 print('clone_list[0]:', clone_list[0])
7 nlis[0] = 'hello, python!'
8 print('nlis[0]:', nlis[0])

```

```
['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]
```

```
['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3, 14, 2022)]
```

```
clone_list[0]: python
```

```
nlis[0]: hello, python!
```

Concatenate the list

In [78]:

```

1 a_list = ['a', 'b', ['c', 'd'], 'e']
2 b_list = [1,2,3,4,5,(6,7), True, False]
3 new_list = a_list + b_list
4 print(new_list)

```

```
['a', 'b', ['c', 'd'], 'e', 1, 2, 3, 4, 5, (6, 7), True, False]
```

As different from the list, I also find significant the following information.

input() function

- **input()** function in Python provides a user of a program supply inputs to the program at runtime.

In [6]:

```

1 text = input('Enter a string:')
2 print('The text is', text)
3 print(type(text))

```

```
The text is Hello, Python!
```

```
<class 'str'>
```

In [12]:

```

1 # Although the function wants an integer, the type of the entered number is a string.
2 number = input('Enter an integer: ')
3 print('The number is', number)
4 print(type(number))

```

```
The number is 15
```

```
<class 'str'>
```

In [15]:

```
1 number = int(input('Enter an integer:'))
2 print('The number is', number)
3 print(type(number))
```

The number is 15
<class 'int'>

In [16]:

```
1 number = float(input('Enter an integer:'))
2 print('The number is', number)
3 print(type(number))
```

The number is 15.0
<class 'float'>

eval() functions

- This function serves the aim of converting a string to an integer or a float

In [17]:

```
1 expression = '8+7'
2 total = eval(expression)
3 print('Sum of the expression is', total)
4 print(type(expression))
5 print(type(total))
```

Sum of the expression is 15
<class 'str'>
<class 'int'>

format() function

- This function helps to format the output printed on the screen with good look and attractive.

In [22]:

```
1 a = float(input('Enter the pi number:'))
2 b = float(input('Enter the golden ratio:'))
3 total = a + b
4 print('Sum of {} and {} is {}'.format(a, b, total))
```

Sum of 3.14 and 1.618 is 4.758.

In [25]:

```
1 a = input('Enter your favorite fruit:')
2 b = input('Enter your favorite food:')
3 print('I like {} and {}'.format(a, b))
4 print('I like {0} and {1}'.format(a, b))
5 print('I like {1} and {0}'.format(a, b))
```

I like apple and kebab.

I like apple and kebab.

I like kebab and apple.

Comparison operators

- The operators such as `<`, `>`, `<=`, `>=`, `==`, and `!=` compare the certain two operands and return *True* or *False*.

In [27]:

```
1 a = 3.14
2 b = 1.618
3 print('a>b is:', a>b)
4 print('a<b is:', a<b)
5 print('a<=b is:', a<=b)
6 print('a>=b is:', a>=b)
7 print('a==b is:', a==b)
8 print('a!=b is:', a!=b)
```

a>b is: True

a<b is: False

a<=b is: False

a>=b is: True

a==b is: False

a!=b is: True

Logical operators

- The operators including **and**, **or**, **not** are utilized to bring two conditions together and assess them. The output returns *True* or *False*

In [35]:

```
1 a = 3.14
2 b = 1.618
3 c = 12
4 d = 3.14
5 print(a>b and c>a)
6 print(b>c and d>a)
7 print(b<c or d>a)
8 print(not a==b)
9 print(not a==d)
```

True

False

True

True

False

Assignment operators

- The operators including =, +=, -=, *=, /=, %=, //=, **=, &=, |=, ^=, >>=, and <<= are employed to evaluate a value to a variable.

In [42]:

```
1 x = 3.14
2 x+=5
3 print(x)
```

8.14

In [43]:

```
1 x = 3.14
2 x-=5
3 print(x)
```

-1.8599999999999999

In [44]:

```
1 x = 3.14
2 x*=5
3 print(x)
```

15.700000000000001

In [45]:

```
1 x = 3.14
2 x/=5
3 print(x)
```

0.628

In [46]:

```
1 x = 3.14
2 x%=5
3 print(x)
```

3.14

In [47]:

```
1 x = 3.14
2 x//=5
3 print(x)
```

0.0

In [48]:

```

1 x = 3.14
2 x**=5
3 print(x)

```

305.2447761824001

Identity operators

- The operators **is** or **is not** are employed to control if the operands or objects to the left and right of these operators are referring to a value stored in the same memory location and return *True* or *False*.

In [74]:

```

1 a = 3.14
2 b = 1.618
3 print(a is b)
4 print(a is not b)
5 msg1= 'Hello, Python!'
6 msg2 = 'Hello, World!'
7 print(msg1 is msg2)
8 print(msg1 is not msg2)
9 lis1 = [3.14, 1.618]
10 lis2 = [3.14, 1.618]
11 print(lis1 is lis2)      # You should see a list copy behavior
12 print(lis1 is not lis2)

```

False
 True
 False
 True
 False
 True

Membership operators

- These operators including **in** and **not in** are employed to check if the certain value is available in the sequence of values and return *True* or *False*.

In [79]:

```

1 # take a list
2 nlis = [4, 6, 7, 8, 'hello', (4,5), {'name': 'Python'}, {1,2,3}, [1,2,3]]
3 print(5 in nlis)
4 print(4 not in nlis)
5 print((4,5) in nlis)
6 print(9 not in nlis)

```

False
 False
 True
 True

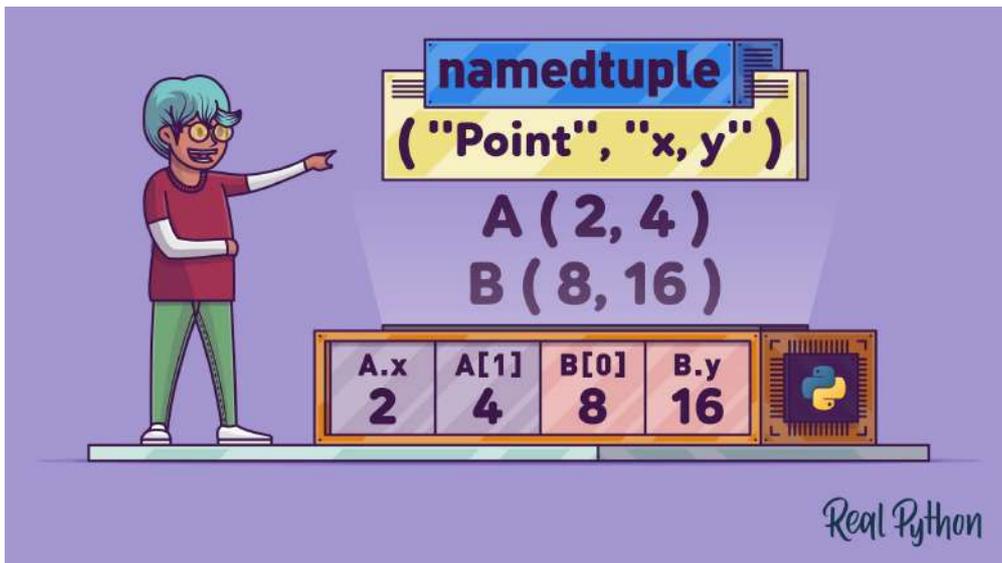
Python Tutorial

Created by Mustafa Germec, PhD

4. Tuples in Python

Tuples are immutable lists and cannot be changed in any way once it is created.

- Tuples are defined in the same way as lists.
- They are enclosed within parenthesis and not within square braces.
- Tuples are ordered, indexed collections of data.
- Similar to string indices, the first value in the tuple will have the index [0], the second value [1]
- Negative indices are counted from the end of the tuple, just like lists.
- Tuple also has the same structure where commas separate the values.
- Tuples can store duplicate values.
- Tuples allow you to store several data items including string, integer, float in one variable.



In [9]:

```
1 # Take a tuple
2 tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
3 tuple_1
```

Out[9]:

```
('Hello',
'Python',
3.14,
1.618,
True,
False,
32,
[1, 2, 3],
{1, 2, 3},
{'A': 3, 'B': 8},
(0, 1))
```

In [10]:

```
1 print(type(tuple_1))
2 print(len(tuple_1))
```

```
<class 'tuple'>
11
```

Indexing

In [12]:

```
1 # Printing the each value in a tuple using both positive and negative indexing
2 tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
3 print(tuple_1[0])
4 print(tuple_1[1])
5 print(tuple_1[2])
6 print(tuple_1[-1])
7 print(tuple_1[-2])
8 print(tuple_1[-3])
```

```
Hello
Python
3.14
(0, 1)
{'A': 3, 'B': 8}
{1, 2, 3}
```

In [11]:

```
1 # Printing the type of each value in the tuple
2 tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
3 print(type(tuple_1[0]))
4 print(type(tuple_1[2]))
5 print(type(tuple_1[4]))
6 print(type(tuple_1[6]))
7 print(type(tuple_1[7]))
8 print(type(tuple_1[8]))
9 print(type(tuple_1[9]))
10 print(type(tuple_1[10]))
```

```
<class 'str'>
<class 'float'>
<class 'bool'>
<class 'int'>
<class 'list'>
<class 'set'>
<class 'dict'>
<class 'tuple'>
```

Concatenation of tuples

To concatenate tuples, + sign is used

In [13]:

```
1 tuple_2 = tuple_1 + ('Hello World!', 2022)
2 tuple_2
```

Out[13]:

```
('Hello',
'Python',
3.14,
1.618,
True,
False,
32,
[1, 2, 3],
{1, 2, 3},
{'A': 3, 'B': 8},
(0, 1),
'Hello World!',
2022)
```

Repetition of a tuple

In [48]:

```
1 rep_tup = (1,2,3,4)
2 rep_tup*2
```

Out[48]:

```
(1, 2, 3, 4, 1, 2, 3, 4)
```

Membership

In [49]:

```
1 rep_tup = (1,2,3,4)
2 print(2 in rep_tup)
3 print(2 not in rep_tup)
4 print(5 in rep_tup)
5 print(5 not in rep_tup)
6
```

```
True
False
False
True
```

Iteration

In [50]:

```
1 rep_tup = (1,2,3,4)
2 for i in rep_tup:
3     print(i)
```

```
1
2
3
4
```

cmp() function

It is to compare two tuples and returns *True* or *False*

In [55]:

```
1 def cmp(t1, t2):
2     return bool(t1 > t2) - bool(t1 < t2)
3 def cmp(t3, t4):
4     return bool(t3 > t4) - bool(t3 < t4)
5 def cmp(t5, t6):
6     return bool(t5 > t6) - bool(t5 < t6)
7 t1 = (1,3,5)      # Here t1 is lower than t2, since the output is -1
8 t2 = (2,4,6)
9
10 t3 = (5,)        # Here t3 is higher than t4 since the output is 1
11 t4 = (4,)
12
13 t5 = (3.14,)     # Here t5 is equal to t6 since the output is 0
14 t6 = (3.14,)
15
16 print(cmp(t1, t2))
17 print(cmp(t3, t4))
18 print(cmp(t5, t6))
```

```
-1
1
0
```

min() function

In [56]:

```
1 rep_tup = (1,2,3,4)
2 min(rep_tup)
```

Out[56]:

```
1
```

max() function

In [58]:

```
1 rep_tup = (1,2,3,4)
2 max(rep_tup)
```

Out[58]:

4

tup(seq) function

It converts a specific sequence to a tuple

In [60]:

```
1 seq = 'ATGCGTATTGCCAT'
2 tuple(seq)
```

Out[60]:

```
('A', 'T', 'G', 'C', 'G', 'T', 'A', 'T', 'T', 'G', 'C', 'C', 'A', 'T')
```

Slicing

To obtain a new tuple from the current tuple, the slicing method is used.

In [14]:

```
1 # Obtaining a new tuple from the index 2 to index 6
2
3 tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
4 tuple_1[2:7]
```

Out[14]:

```
(3.14, 1.618, True, False, 32)
```

In [18]:

```
1 # Obtaining tuple using negative indexing
2 tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
3 tuple_1[-4:-1]
```

Out[18]:

```
([1, 2, 3], {1, 2, 3}, {'A': 3, 'B': 8})
```

len() function

To obtain how many elements there are in the tuple, use len() function.

In [19]:

```
1 tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
2 len(tuple_1)
```

Out[19]:

11

Sorting tuple

In [22]:

```
1 # Tuples can be sorted and save as a new tuple.
2
3 tuple_3 = (0,9,7,4,6,2,9,8,3,1)
4 sorted_tuple_3 = sorted(tuple_3)
5 sorted_tuple_3
```

Out[22]:

[0, 1, 2, 3, 4, 6, 7, 8, 9, 9]

Nested tuple

In Python, a tuple written inside another tuple is known as a nested tuple.

In [25]:

```
1 # Take a nested tuple
2 nested_tuple = ('biotechnology', (0, 5), ('fermentation', 'ethanol'), (3.14, 'pi', (1.618, 'golden ratio')))
3 nested_tuple
```

Out[25]:

```
('biotechnology',
(0, 5),
('fermentation', 'ethanol'),
(3.14, 'pi', (1.618, 'golden ratio')))
```

In [26]:

```
1 # Now printing the each element of the nested tuple
2 print('Item 0 of nested tuple is', nested_tuple[0])
3 print('Item 1 of nested tuple is', nested_tuple[1])
4 print('Item 2 of nested tuple is', nested_tuple[2])
5 print('Item 3 of nested tuple is', nested_tuple[3])
```

Element 0 of nested tuple is biotechnology

Element 1 of nested tuple is (0, 5)

Element 2 of nested tuple is ('fermentation', 'ethanol')

Element 3 of nested tuple is (3.14, 'pi', (1.618, 'golden ratio'))

In [33]:

```

1 # Using second index to access other tuples in the nested tuple
2 print('Item 1, 0 of the nested tuple is', nested_tuple[1][0])
3 print('Item 1, 1 of the nested tuple is', nested_tuple[1][1])
4 print('Item 2, 0 of the nested tuple is', nested_tuple[2][0])
5 print('Item 2, 1 of the nested tuple is', nested_tuple[2][1])
6 print('Item 3, 0 of the nested tuple is', nested_tuple[3][0])
7 print('Item 3, 1 of the nested tuple is', nested_tuple[3][1])
8 print('Item 3, 2 of the nested tuple is', nested_tuple[3][2])
9
10 # Accesing to the items in the second nested tuples using a third index
11 print('Item 3, 2, 0 of the nested tuple is', nested_tuple[3][2][0])
12 print('Item 3, 2, 1 of the nested tuple is', nested_tuple[3][2][1])

```

Item 1, 0 of the nested tuple is 0
 Item 1, 1 of the nested tuple is 5
 Item 2, 0 of the nested tuple is fermentation
 Item 2, 1 of the nested tuple is ethanol
 Item 3, 0 of the nested tuple is 3.14
 Item 3, 1 of the nested tuple is pi
 Item 3, 2 of the nested tuple is (1.618, 'golden ratio')
 Item 3, 2, 0 of the nested tuple is 1.618
 Item 3, 2, 1 of the nested tuple is golden ratio

Tuples are immutable

In [35]:

```

1 # Take a tuple
2 tuple_4 = (1,3,5,7,8)
3 tuple_4[0] = 9
4 print(tuple_4)
5
6 # The output shows the tuple is immutable

```

```

-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17624\4165256041.py in <module>
      1 # Take a tuple
      2 tuple_4 = (1,3,5,7,8)
----> 3 tuple_4[0] = 9
      4 print(tuple_4)
      5

```

TypeError: 'tuple' object does not support item assignment

Delete a tuple

- An element in a tuple can not be deleted since it is immutable.
- But a whole tuple can be deleted

In [36]:

```
1 tuple_4 = (1,3,5,7,8)
2 print('Before deleting:', tuple_4)
3 del tuple_4
4 print('After deleting:', tuple_4)
```

Before deleting: (1, 3, 5, 7, 8)

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_17624\736020228.py in <module>
      2 print('Before deleting:', tuple_4)
      3 del tuple_4
----> 4 print('After deleting:', tuple_4)
```

NameError: name 'tuple_4' is not defined

count() method

This method returns the number of time an item occurs in a tuple.

In [39]:

```
1 tuple_5 = (1,1,3,3,5,5,5,5,6,6,7,8,9)
2 tuple_5.count(5)
```

Out[39]:

4

index() method

It returns the index of the first occurrence of the specified value in a tuple

In [42]:

```
1 tuple_5 = (1,1,3,3,5,5,5,5,6,6,7,8,9)
2 print(tuple_5.index(5))
3 print(tuple_5.index(1))
4 print(tuple_5.index(9))
```

```
4
0
12
```

One element tuple

if a tuple includes only one element, you should put a comma after the element. Otherwise, it is not considered as a tuple.

In [45]:

```
1 tuple_6 = (0)
2 print(tuple_6)
3 print(type(tuple_6))
4
5 # Here, you see that the output is an integer
```

```
0
<class 'int'>
```

In [47]:

```
1 tuple_7 = (0,)
2 print(tuple_7)
3 print(type(tuple_7))
4
5 # You see that the output is a tuple
```

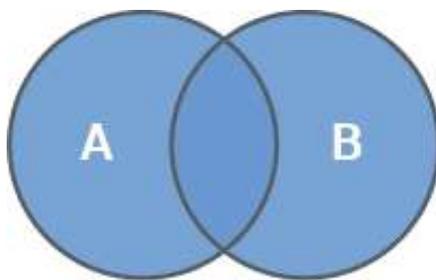
```
(0,)
<class 'tuple'>
```

Python Tutorial

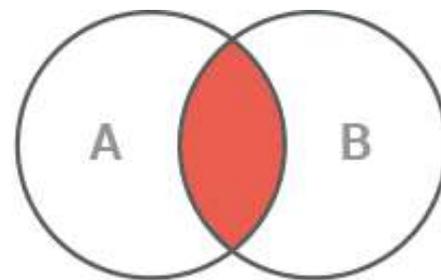
Created by Mustafa Germec, PhD

5. Sets in Python

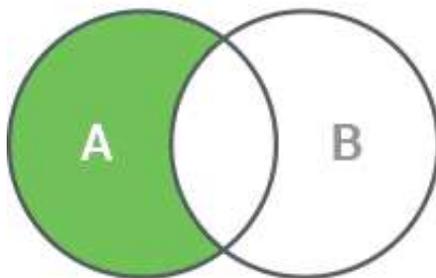
- Set is one of 4 built-in data types in Python used to store collections of data including List, Tuple, and Dictionary
- Sets are unordered, but you can remove items and add new items.
- Set elements are unique. Duplicate elements are not allowed.
- A set itself may be modified, but the elements contained in the set must be of an immutable type.
- Sets are used to store multiple items in a single variable.
- You can denote a set with a pair of curly brackets `{}`.



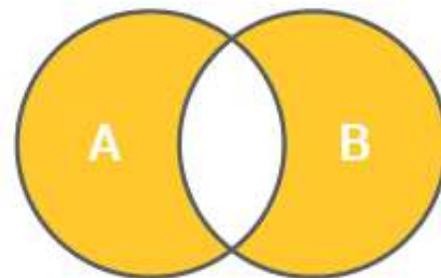
Union



Intersection



Difference



Symmetric Difference

In [47]:

```
1 # The empty set of curly braces denotes the empty dictionary, not empty set
2 x = {}
3 print(type(x))
```

<class 'dict'>

In [46]:

```
1 # To take a set without elements, use set() function without any items
2 y = set()
3 print(type(y))
```

<class 'set'>

In [2]:

```
1 # Take a set
2 set1 = {'Hello Python!', 3.14, 1.618, 'Hello World!', 3.14, 1.618, True, False, 2022}
3 set1
```

Out[2]:

```
{1.618, 2022, 3.14, False, 'Hello Python!', 'Hello World!', True}
```

Converting list to set

In [4]:

```
1 # A list can convert to a set
2 # Take a list
3 nlis = ['Hello Python!', 3.14, 1.618, 'Hello World!', 3.14, 1.618, True, False, 2022]
4
5 # Convert the list to a set
6 set2 = set(nlis)
7 set2
```

Out[4]:

```
{1.618, 2022, 3.14, False, 'Hello Python!', 'Hello World!', True}
```

Set operations

In [5]:

```
1 # Take a set
2 set3 = set(['Hello Python!', 3.14, 1.618, 'Hello World!', 3.14, 1.618, True, False, 2022])
3 set3
```

Out[5]:

```
{1.618, 2022, 3.14, False, 'Hello Python!', 'Hello World!', True}
```

add() function

To add an element into a set, we use the function **add()**. If the same element is added to the set, nothing will happen because the set accepts no duplicates.

In [6]:

```
1 # Addition of an element to a set
2 set3 = set(['Hello Python!', 3.14, 1.618, 'Hello World!', 3.14, 1.618, True, False, 2022])
3 set3.add('Hi, Python!')
4 set3
```

Out[6]:

```
{1.618,
2022,
3.14,
False,
'Hello Python!',
'Hello World!',
'Hi, Python!',
True}
```

In [7]:

```
1 # Addition of the same element
2 set3.add('Hi, Python!')
3 set3
4
5 # As you see that there is only one from the added element 'Hi, Python!'
```

Out[7]:

```
{1.618,
2022,
3.14,
False,
'Hello Python!',
'Hello World!',
'Hi, Python!',
True}
```

update() function

To add multiple elements into the set

In [49]:

```
1 x_set = {6,7,8,9}
2 print(x_set)
3 x_set.update({3,4,5})
4 print(x_set)
```

```
{8, 9, 6, 7}
{3, 4, 5, 6, 7, 8, 9}
```

remove() function

To **remove** an element from the set

In [16]:

```
1 set3.remove('Hello Python!')
2 set3
3
```

Out[16]:

```
{1.618, 2022, 3.14, False, 'Hello World!', True}
```

discard() function

It leaves the set unchanged if the element to be deleted is not available in the set.

In [50]:

```
1 set3.discard(3.14)
2 set3
```

Out[50]:

```
{1.618, 2022, False, 'Hello World!', True}
```

In [17]:

```
1 # To verify if the element is in the set
2 1.618 in set3
```

Out[17]:

```
True
```

Logic operations in Sets

In [18]:

```
1 # Take two sets
2 set4 = set(['Hello Python!', 3.14, 1.618, 'Hello World!'])
3 set5 = set([3.14, 1.618, True, False, 2022])
4
5 # Printing two sets
6 set4, set5
```

Out[18]:

```
{1.618, 3.14, 'Hello Python!', 'Hello World!'},
{False, True, 1.618, 3.14, 2022}
```

To find the intersect of two sets using &

In [19]:

```
1 intersection = set4 & set5
2 intersection
```

Out[19]:

```
{1.618, 3.14}
```

To find the intersect of two sets, use intersection() function

In [21]:

```
1 set4.intersection(set5) # The output is the same as that of above
```

Out[21]:

```
{1.618, 3.14}
```

difference() function

To find the difference between two sets

In [61]:

```
1 print(set4.difference(set5))
2 print(set5.difference(set4))
3
4 # The same process can make using subtraction operator as follows:
5 print(set4-set5)
6 print(set5-set4)
```

```
{'Hello Python!', 'Hello World!'}
```

```
{False, True, 2022}
```

```
{'Hello Python!', 'Hello World!'}
```

```
{False, True, 2022}
```

Set comparison

In [62]:

```
1 print(set4>set5)
2 print(set5>set4)
3 print(set4==set5)
```

```
False
```

```
False
```

```
False
```

union() function

it corresponds to all the elements in both sets

In [24]:

```
1 set4.union(set5)
```

Out[24]:

```
{1.618, 2022, 3.14, False, 'Hello Python!', 'Hello World!', True}
```

issuperset() and issubset() functions

To control if a set is a superset or a subset of another set

In [25]:

```
1 set(set4).issuperset(set5)
```

Out[25]:

```
False
```

In [27]:

```
1 set(set4).issubset(set5)
```

Out[27]:

```
False
```

In [34]:

```
1 print(set([3.14, 1.618]).issubset(set5))
2 print(set([3.14, 1.618]).issubset(set4))
3 print(set4.issuperset([3.14, 1.618]))
4 print(set5.issuperset([3.14, 1.618]))
```

```
True
```

```
True
```

```
True
```

```
True
```

min(), max() and sum() functions

In [36]:

```

1 A = [1,1,2,2,3,3,4,4,5,5] # Take a list
2 B = {1,1,2,2,3,3,4,4,5,5} # Take a set
3
4 print('The minimum number of A is', min(A))
5 print('The minimum number of B is', min(B))
6 print('The maximum number of A is', max(A))
7 print('The maximum number of B is', max(B))
8 print('The sum of A is', sum(A))
9 print('The sum of B is', sum(B))
10
11 # As you see that the sum of A and B is different. Because the set takes no duplicate.

```

The minimum number of A is 1

The minimum number of B is 1

The maximum number of A is 5

The maximum number of B is 5

The sum of A is 30

The sum of B is 15

No mutable sequence in a set

A set can not have mutable elements such as list or dictionary in it. If any, it returns error as follows:

In [39]:

```

1 set6 = {'Python', 1,2,3, [1,2,3]}
2 set6

```

```

-----
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_10540\2974310107.py in <module>
----> 1 set6 = {'Python', 1,2,3, [1,2,3]}
      2 set6

```

TypeError: unhashable type: 'list'

index() function

This function does not work in set since the set is unordered collection

In [48]:

```

1 set7 = {1,2,3,4}
2 set7[1]

```

```

-----
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_10540\893084458.py in <module>
      1 set7 = {1,2,3,4}
----> 2 set7[1]

```

TypeError: 'set' object is not subscriptable

Copy the set

In [54]:

```

1 set8 = {1,3,5,7,9}
2 print(set8)
3 set9 = set8
4 print(set9)
5 set8.add(11)
6 print(set8)
7 print(set9)
8
9 """
10 As you see that although the number 8 is added into the set 'set8', the added number
11 is also added into the set 'set9'
12 """

```

```

{1, 3, 5, 7, 9}
{1, 3, 5, 7, 9}
{1, 3, 5, 7, 9, 11}
{1, 3, 5, 7, 9, 11}

```

copy() function

it returns a shallow copy of the original set.

In [56]:

```

1 set8 = {1,3,5,7,9}
2 print(set8)
3 set9 = set8.copy()
4 print(set9)
5 set8.add(11)
6 print(set8)
7 print(set9)
8
9 """
10 When this function is used, the original set stays unmodified.
11 A new copy stored in another set of memory locations is created.
12 The change made in one copy won't reflect in another.
13 """

```

```

{1, 3, 5, 7, 9}
{1, 3, 5, 7, 9}
{1, 3, 5, 7, 9, 11}
{1, 3, 5, 7, 9}

```

Out[56]:

```

"\nWhen this function is used, the original set stays unmodified.\nA new copy stored in another set of
memory locations is created.\nThe change made in one copy won't reflect in another.\n"

```

clear() function

it removes all elements in the set and then do the set empty.

In [57]:

```
1 x = {0, 1,1,2,3,5,8,13, 21,34}
2 print( x)
3 x.clear()
4 print(x)
```

```
{0, 1, 2, 3, 34, 5, 8, 13, 21}
set()
```

pop() function

It removes and returns an arbitrary set element.

In [60]:

```
1 x = {0, 1,1,2,3,5,8,13,21,34}
2 print(x)
3 x.pop()
4 print(x)
```

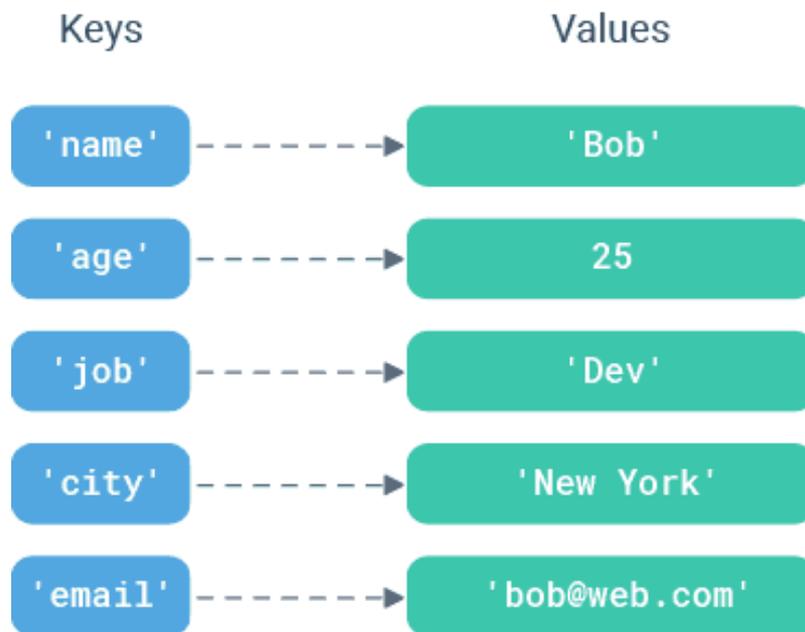
```
{0, 1, 2, 3, 34, 5, 8, 13, 21}
{1, 2, 3, 34, 5, 8, 13, 21}
```

Python Tutorial

Created by Mustafa Germec, PhD

6. Dictionaries in Python

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered, changeable or mutable and do not allow duplicates.
- Dictionary items are ordered, changeable, and does not allow duplicates.
- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.
- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
- Dictionaries cannot have two items with the same key.
- A dictionary can nested and can contain another dictionary.



In [1]:

```
1 # Take a sample dictionary
2
3 sample_dict = {'key_1': 3.14, 'key_2': 1.618,
4               'key_3': True, 'key_4': [3.14, 1.618],
5               'key_5': (3.14, 1.618), 'key_6': 2022, (3.14, 1.618): 'pi and golden ratio'}
6 sample_dict
```

Out[1]:

```
{'key_1': 3.14,
'key_2': 1.618,
'key_3': True,
'key_4': [3.14, 1.618],
'key_5': (3.14, 1.618),
'key_6': 2022,
(3.14, 1.618): 'pi and golden ratio'}
```

Note: As you see that the whole dictionary is enclosed in curly braces, each key is separated from its value by a column ":", and commas are used to separate the items in the dictionary.

In [4]:

```

1 # Accessing to the value using the key
2 print(sample_dict['key_1'])
3 print(sample_dict['key_2'])
4 print(sample_dict['key_3'])
5 print(sample_dict['key_4'])
6 print(sample_dict['key_5'])
7 print(sample_dict['key_6'])
8 print(sample_dict[(3.14, 1.618)]) # Keys can be any immutable object like tuple

```

```

3.14
1.618
True
[3.14, 1.618]
(3.14, 1.618)
2022
pi and golden ratio

```

Keys

In [26]:

```

1 # Take a sample dictionary
2 product = {'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol',
3           'Scheffersomyces stipitis': 'ethanol', 'Aspergillus sojae_1': 'mannanase',
4           'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid',
5           'Aspergillus sojae_2': 'polygalacturonase'}
6 product
7

```

Out[26]:

```

{'Aspergillus niger': 'inulinase',
'Saccharomyces cerevisiae': 'ethanol',
'Scheffersomyces stipitis': 'ethanol',
'Aspergillus sojae_1': 'mannanase',
'Streptococcus zooepidemicus': 'hyaluronic acid',
'Lactobacillus casei': 'lactic acid',
'Aspergillus sojae_2': 'polygalacturonase'}

```

In [27]:

```

1 # Retrieving the value by keys
2 print(product['Aspergillus niger'])
3 print(product['Saccharomyces cerevisiae'])
4 print(product['Scheffersomyces stipitis'])

```

```

inulinase
ethanol
ethanol

```

keys() function to get the keys in the dictionary

In [28]:

```
1 # What are the keys in the dictionary?
2 product.keys()
```

Out[28]:

```
dict_keys(['Aspergillus niger', 'Saccharomyces cerevisiae', 'Scheffersomyces stipitis', 'Aspergillus sojae_1', 'Streptococcus zooepidemicus', 'Lactobacillus casei', 'Aspergillus sojae_2'])
```

values() function to get the values in the dictionary

In [29]:

```
1 # What are the values in the dictionary?
2 product.values()
```

Out[29]:

```
dict_values(['inulinase', 'ethanol', 'ethanol', 'mannanase', 'hyaluronic acid', 'lactic acid', 'polygalacturona se'])
```

Addition of a new key:value pair in the dictionary

In [31]:

```
1 product['Yarrowia lipolytica'] = 'microbial oil'
2 product
```

Out[31]:

```
{'Aspergillus niger': 'inulinase',
'Saccharomyces cerevisiae': 'ethanol',
'Scheffersomyces stipitis': 'ethanol',
'Aspergillus sojae_1': 'mannanase',
'Streptococcus zooepidemicus': 'hyaluronic acid',
'Lactobacillus casei': 'lactic acid',
'Aspergillus sojae_2': 'polygalacturonase',
'Yarrowia lipolytica': 'microbial oil'}
```

Delete an item using del() function in the dictionary by key

In [32]:

```
1 del(product['Aspergillus niger'])
2 del(product['Aspergillus sojae_1'])
3 product
```

Out[32]:

```
{'Saccharomyces cerevisiae': 'ethanol',
'Scheffersomyces stipitis': 'ethanol',
'Streptococcus zooepidemicus': 'hyaluronic acid',
'Lactobacillus casei': 'lactic acid',
'Aspergillus sojae_2': 'polygalacturonase',
'Yarrowia lipolytica': 'microbial oil'}
```

In [1]:

```

1 del product
2 print(product)
3
4 # The dictionary was deleted.
```

```

-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_2904\1117454704.py in <module>
----> 1 del product
      2 print(product)
      3
      4 # The dictionary was deleted.
```

NameError: name 'product' is not defined

Verification using in or not in

In [17]:

```

1 print('Saccharomyces cerevisiae' in product)
2 print('Saccharomyces cerevisiae' not in product)
```

True
False

dict() function

This function is used to create a dictionary

In [19]:

```

1 dict_sample = dict(family = 'music', type='pop', year='2022', name='happy new year')
2 dict_sample
```

Out[19]:

```
{'family': 'music', 'type': 'pop', 'year': '2022', 'name': 'happy new year'}
```

In [21]:

```

1 # Numerical index is not used to take the dictionary values. It gives a KeyError
2 dict_sample[1]
```

```

-----
KeyError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_3576\4263495629.py in <module>
      1 # Numerical index is not used to take the dictionary values. It gives a KeyError
----> 2 dict_sample[1]
```

KeyError: 1

clear() functions

It removes all the items in the dictionary and returns an empty dictionary

In [34]:

```
1 dict_sample = dict(family = 'music', type='pop', year='2022', name='happy new year')
2 dict_sample.clear()
3 dict_sample
```

Out[34]:

```
{}
```

copy() function

It returns a shallow copy of the main dictionary

In [35]:

```
1 sample_original = dict(family = 'music', type='pop', year='2022', name='happy new year')
2 sample_copy = sample_original.copy()
3 print(sample_original)
4 print(sample_copy)
```

```
{'family': 'music', 'type': 'pop', 'year': '2022', 'name': 'happy new year'}
```

```
{'family': 'music', 'type': 'pop', 'year': '2022', 'name': 'happy new year'}
```

In [36]:

```
1 # This method can be made usign '=' sign
2 sample_copy = sample_original
3 print(sample_copy)
4 print(sample_original)
```

```
{'family': 'music', 'type': 'pop', 'year': '2022', 'name': 'happy new year'}
```

```
{'family': 'music', 'type': 'pop', 'year': '2022', 'name': 'happy new year'}
```

pop() function

This function is used to remove a specific item from the dictionary

In [38]:

```
1 sample_original = dict(family = 'music', type='pop', year='2022', name='happy new year')
2 print(sample_original.pop('type'))
3 print(sample_original)
4
```

```
pop
```

```
{'family': 'music', 'year': '2022', 'name': 'happy new year'}
```

popitem() function

It is used to remove the **arbitrary** items from the dictionary and returns as a tuple.

In [39]:

```
1 sample_original = dict(family = 'music', type='pop', year='2022' , name='happy new year')
2 print(sample_original.popitem())
3 print(sample_original)
```

```
('name', 'happy new year')
{'family': 'music', 'type': 'pop', 'year': '2022'}
```

get() function

This method returns the value for the specified key if it is available in the dictionary. If the key is not available, it returns *None*.

In [41]:

```
1 sample_original = dict(family = 'music', type='pop', year='2022' , name='happy new year')
2 print(sample_original.get('family'))
3 print(sample_original.get(3))
```

```
music
None
```

fromkeys() function

It returns a new dictionary with the certain sequence of the items as the keys of the dictionary and the values are assigned with *None*.

In [44]:

```
1 keys = {'A', 'T', 'C', 'G'}
2 sequence = dict.fromkeys(keys)
3 print(sequence)
```

```
{'C': None, 'T': None, 'A': None, 'G': None}
```

update() function

It integrates a dictionary with another dictionary or with an iterable of key:value pairs.

In [45]:

```
1 product = {'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol',
2           'Scheffersomyces stipitis': 'ethanol', 'Aspergillus sojae_1': 'mannanase',
3           'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid',
4           'Aspergillus sojae_2': 'polygalacturonase'}
5
6 sample_original = dict(family = 'music', type='pop', year='2022', name='happy new year')
7
8 product.update(sample_original)
9 print(product)
```

```
{'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol', 'Scheffersomyces stipitis': 'ethanol',
'Aspergillus sojae_1': 'mannanase', 'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid', 'Aspergillus sojae_2': 'polygalacturonase', 'family': 'music', 'type': 'pop', 'year': '2022', 'name': 'happy new year'}
```

items() function

It returns a list of key:value pairs in a dictionary. The elements in the lists are tuples.

In [46]:

```
1 product = {'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol',
2           'Scheffersomyces stipitis': 'ethanol', 'Aspergillus sojae_1': 'mannanase',
3           'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid',
4           'Aspergillus sojae_2': 'polygalacturonase'}
5
6 product.items()
```

Out[46]:

```
dict_items([('Aspergillus niger', 'inulinase'), ('Saccharomyces cerevisiae', 'ethanol'), ('Scheffersomyces stipitis', 'ethanol'), ('Aspergillus sojae_1', 'mannanase'), ('Streptococcus zooepidemicus', 'hyaluronic acid'), ('Lactobacillus casei', 'lactic acid'), ('Aspergillus sojae_2', 'polygalacturonase')])
```

Iterating dictionary

A dictionary can be iterated using the for loop

In [11]:

```
1 # 'for' loop print all the keys in the dictionary
2
3 product = {'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol',
4            'Scheffersomyces stipitis': 'ethanol', 'Aspergillus sojae_1': 'mannanase',
5            'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid',
6            'Aspergillus sojae_2': 'polygalacturonase'}
7
8 for k in product:
9     print(k)
```

Aspergillus niger
Saccharomyces cerevisiae
Scheffersomyces stipitis
Aspergillus sojae_1
Streptococcus zooepidemicus
Lactobacillus casei
Aspergillus sojae_2

In [15]:

```
1 # 'for' loop to print the values of the dictionary by using values() and other method
2
3 product = {'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol',
4            'Scheffersomyces stipitis': 'ethanol', 'Aspergillus sojae_1': 'mannanase',
5            'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid',
6            'Aspergillus sojae_2': 'polygalacturonase'}
7 for x in product.values():
8     print(x)
9
10 print()
11 # 'for' loop to print the values of the dictionary by using values() and other method
12 for x in product:
13     print(product[x])
```

inulinase
ethanol
ethanol
mannanase
hyaluronic acid
lactic acid
polygalacturonase

inulinase
ethanol
ethanol
mannanase
hyaluronic acid
lactic acid
polygalacturonase

In [16]:

```
1 # 'for' loop to print the items of the dictionary by using items() method
2 product = {'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol',
3           'Scheffersomyces stipitis': 'ethanol', 'Aspergillus sojae_1': 'mannanase',
4           'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid',
5           'Aspergillus sojae_2': 'polygalacturonase'}
6
7 for x in product.items():
8     print(x)
```

```
('Aspergillus niger', 'inulinase')
('Saccharomyces cerevisiae', 'ethanol')
('Scheffersomyces stipitis', 'ethanol')
('Aspergillus sojae_1', 'mannanase')
('Streptococcus zooepidemicus', 'hyaluronic acid')
('Lactobacillus casei', 'lactic acid')
('Aspergillus sojae_2', 'polygalacturonase')
```

In [17]:

```
1 product = {'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol',
2           'Scheffersomyces stipitis': 'ethanol', 'Aspergillus sojae_1': 'mannanase',
3           'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus casei': 'lactic acid',
4           'Aspergillus sojae_2': 'polygalacturonase'}
5
6 for x, y in product.items():
7     print(x, y)
```

```
Aspergillus niger inulinase
Saccharomyces cerevisiae ethanol
Scheffersomyces stipitis ethanol
Aspergillus sojae_1 mannanase
Streptococcus zooepidemicus hyaluronic acid
Lactobacillus casei lactic acid
Aspergillus sojae_2 polygalacturonase
```

Python Tutorial

Created by Mustafa Germec, PhD

7. Conditions in Python

Comparison operators

Comparison operations compare some value or operand and based on a condition, produce a Boolean. Python has six comparison operators as below:

- Less than (<)
- Less than or equal to (<=)
- Greater than (>)
- Greater than or equal to (>=)
- Equal to (==)
- Not equal to (!=)

In [1]:

```
1 # Take a variable
2 golden_ratio = 1.618
3
4 # Condition less than
5 print(golden_ratio<2) # The golden ratio is lower than 2, thus the output is True
6 print(golden_ratio<1) # The golden ratio is greater than 1, thus the output is False
```

True
False

In [4]:

```
1 # Take a variable
2 golden_ratio = 1.618
3
4 # Condition less than or equal to
5 print(golden_ratio<=2) # The golden ratio is lower than 2, thus the condition is True.
6 print(golden_ratio<=1) # The golden ratio is greater than 1, thus the condition is False.
7 print(golden_ratio<=1.618) # The golden ratio is equal to 1.618, thus the condition is True.
```

True
False
True

In [5]:

```
1 # Take a variable
2 golden_ratio = 1.618
3
4 # Condition greater than
5 print(golden_ratio>2) # The golden ratio is lower than 2, thus the condition is False.
6 print(golden_ratio>1) # The golden ratio is greater than 1, thus the condition is True.
```

False

True

In [7]:

```
1 # Take a variable
2 golden_ratio = 1.618
3
4 # Condition greater than or equal to
5 print(golden_ratio>=2) # The golden ratio is not greater than 2, thus the condition is False.
6 print(golden_ratio>=1) # The golden ratio is greater than 1, thus the condition is True.
7 print(golden_ratio>=1.618) # The golden ratio is equal to 1.618, thus the condition is True.
```

False

True

True

In [8]:

```
1 # Take a variable
2 golden_ratio = 1.618
3
4 # Condition equal to
5 print(golden_ratio==2) # The golden ratio is not equal to 1.618, thus the condition is False.
6 print(golden_ratio==1.618) # The golden ratio is equal to 1.618, thus the condition is True.
```

False

True

In [11]:

```
1 # Take a variable
2 golden_ratio = 1.618
3
4 # Condition not equal to
5 print(golden_ratio!=2) # The golden ratio is not equal to 1.618, thus the condition is True.
6 print(golden_ratio!=1.618) # The golden ratio is equal to 1.618, thus the condition is False.
```

True

False

The comparison operators are also employed to compare the letters/words/symbols according to the [ASCII](https://www.asciitable.com/) (<https://www.asciitable.com/>) value of letters.

In [17]:

```

1 # Compare strings
2 print('Hello' == 'Python')
3 print('Hello' != 'Python')
4 print('Hello' <= 'Python')
5 print('Hello' >= 'Python')
6 print('Hello' < 'Python')
7 print('Hello' > 'Python')
8 print('B'>'A') # According to ASCII table, the values of A and B are equal 65 and 66, respectively.
9 print('a'>'b') # According to ASCII table, the values of a and b are equal 97 and 98, respectively.
10 print('CD'>'DC') # According to ASCII table, the value of C (67) is lower than that of D (68)
11
12 # The values of uppercase and lowercase letters are different since python is case sensitive.

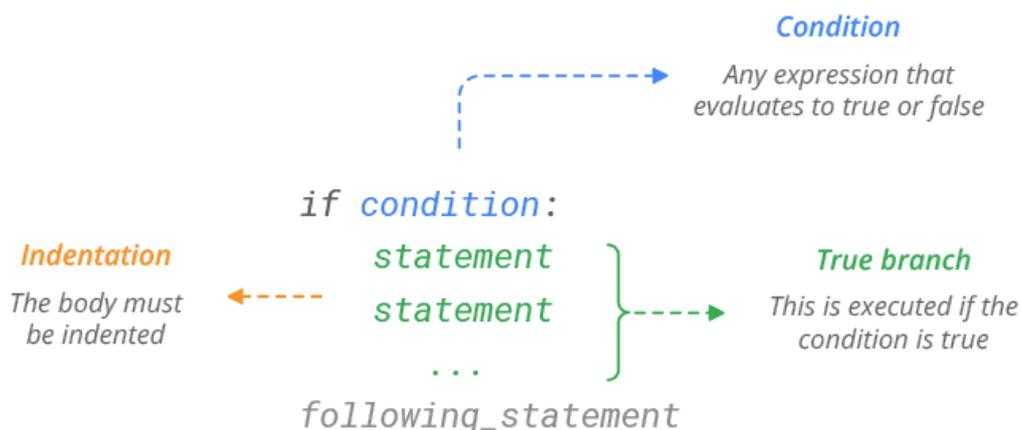
```

False
True
True
False
True
False
True
False
False

Branching (if, elif, else)

- Decision making is required when we want to execute a code only if a certain condition is satisfied.
- The **if/elif/else** statement is used in Python for decision making.
- An **else** statement can be combined with an **if** statement.
- An **else** statement contains the block of code that executes if the conditional expression in the **if** statement resolves to **0** or a **False** value
- The **else** statement is an optional statement and there could be at most only one else statement following **if**.
- The **elif** statement allows you to check multiple expressions for **True** and execute a block of code as soon as one of the conditions evaluates to **True**.
- Similar to the **else**, the **elif** statement is optional.
- However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

If statement



In [6]:

```

1 pi = 3.14
2 golden_ratio = 1.618
3
4 # This statement can be True or False.
5 if pi > golden_ratio:
6
7     # If the conditions is True, the following statement will be printed.
8     print(f'The number pi {pi} is greater than the golden ratio {golden_ratio}.')
9
10 # The following statement will be printed in each situation.
11 print('Done!')
```

The number pi 3.14 is greater than the golden ratio 1.618.
Done!

In [2]:

```

1 if 2:
2     print('Hello, python!')
```

Hello, python!

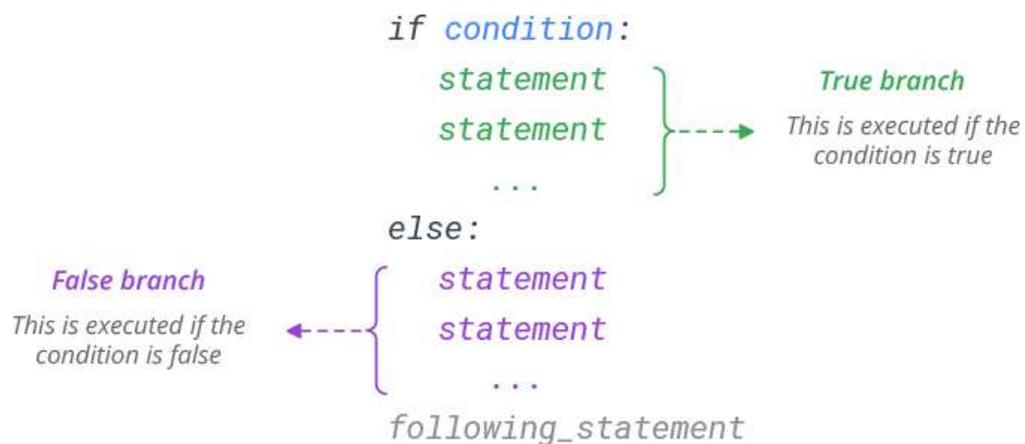
In [5]:

```

1 if True:
2     print('This is true.')
```

This is true.

else statement



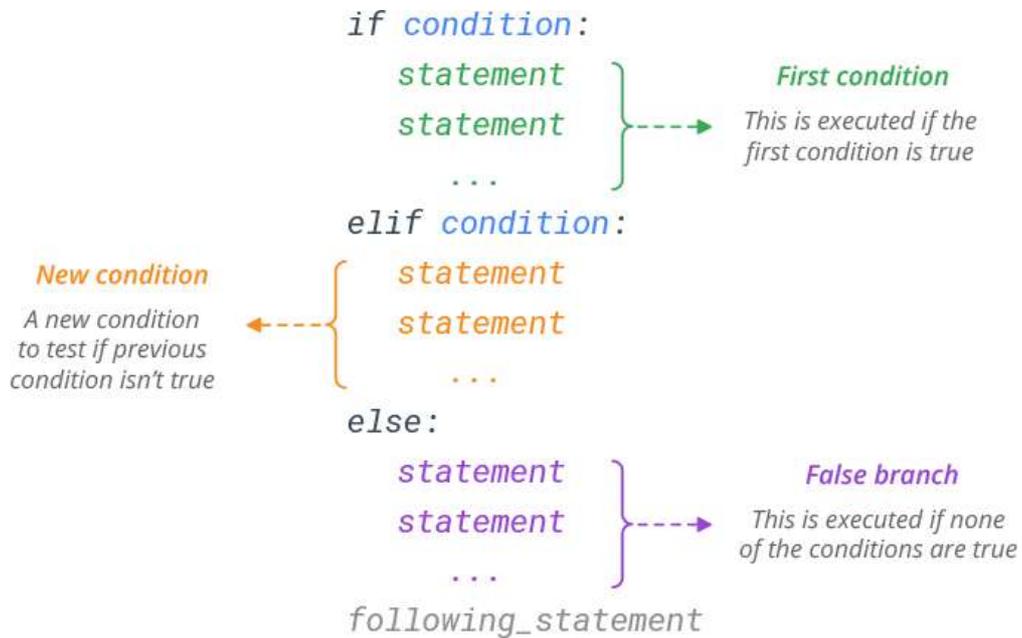
In [8]:

```

1 pi = 3.14
2 golden_ratio = 1.618
3
4 if pi < golden_ratio:
5     print(f'The number pi {pi} is greater than the golden ratio {golden_ratio}.')
6 else:
7     print(f'The golden ratio {golden_ratio} is lower than the number pi {pi}.')
8 print('Done!')
```

The golden ratio 1.618 is lower than the number pi 3.14.
Done!

elif statement



In [23]:

```

1 age = 5
2
3 if age > 6:
4     print('You can go to primary school. ')
5 elif age == 5:
6     print('You should go to kindergarten. ')
7 else:
8     print('You are a baby' )
9
10 print('Done!')
```

You should go to kindergarten.
Done!

In [25]:

```
1 album_year = 2000
2 album_year = 1990
3
4 if album_year >= 1995:
5     print('Album year is higher than 1995.')
6
7 print('Done!')
```

Done!

In [26]:

```
1 album_year = 2000
2 # album_year = 1990
3
4 if album_year >= 1995:
5     print('Album year is higher than 1995.')
6 else:
7     print('Album year is lower than 1995.')
8
9 print('Done!')
```

Album year is higher than 1995.

Done!

In [43]:

```
1 imdb_point = 9.0
2 if imdb_point > 8.5:
3     print('The movie could win Oscar.')
```

The movie could win Oscar.

In [13]:

```
1 movie_rating = float(input('Enter a rating number:'))
2
3 print(f'The entered movie rating is: {movie_rating}')
4
5 if movie_rating > 8.5:
6     print('The movie is awesome with {} rating and you should watch it.'.format(movie_rating))
7 else:
8     print('The movie has merit to be watched with {} rating.'.format(movie_rating))
```

The entered movie rating is: 8.2

The movie has merit to be watched with 8.2 rating.

In [18]:

```
1 note = float(input('Enter a note:'))
2
3 print(f'The entered note value is: {note}')
4
5 if note >= 90 and note <= 100:
6     print('The letter grade is AA.')
7 elif note >= 85 and note <= 89:
8     print('The letter grade is BA.')
9 elif note >= 80 and note <= 84:
10    print('The letter grade is BB.')
11 elif note >= 75 and note <= 79:
12    print('The letter grade is CB.')
13 elif note >= 70 and note <= 74:
14    print('The letter grade is CC.')
15 elif note >= 65 and note <= 69:
16    print('The letter grade is DC.')
17 elif note >= 60 and note <= 64:
18    print('The letter grade is DD.')
19 elif note >=55 and note <= 59:
20    print('The letter grade is ED.')
21 elif note >=50 and note <= 54:
22    print('The letter grade is EE.')
23 elif note >=45 and note <=49:
24    print('The letter grade is FE.')
25 else:
26    print('The letter grade is FF.')
```

The entered note value is: 74.0
The letter grade is CC.

In [17]:

```
1 number = int(input('Enter a number:'))
2
3 print(f'The entered number is: {number}')
4
5 if number %2 == 0:
6     print(f'The entered number {number} is even')
7 else:
8     print(f'The entered number {number} is odd')
```

The entered number is 12
The entered number 12 is even

Logical operators

Logical operators are used to combine conditional statements.

- **and:** Returns True if both statements are true
- **or:** Returns True if one of the statements is true
- **not:** Reverse the result, returns False if the result is true

Python Logical Operators

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

A	Not A
True	False
False	True

and

In [27]:

```
1 birth_year = 1990
2 if birth_year > 1989 and birth_year < 1995:
3     print('You were born between 1990 and 1994')
4 print('Done!')
```

You were born between 1990 and 1994

Done!

In [23]:

```
1 x = int(input('Enter a number:'))
2 y = int(input('Enter a number: '))
3 z = int(input('Enter a number:'))
4
5 print(f'The entered numbers for x, y, and z are {x}, {y}, and {z}, respectively.')
6
7 if x>y and x>z:
8     print(f'The number x with {x} is the greatest number.')
9 elif y>x and y>z:
10    print(f'The number y with {y} is the greatest number.')
11 else:
12    print(f'The number z with {z} is the greatest number.')
```

The entered numbers for x, y, and z are 36, 25, and 21, respectively.

The number x with 36 is the greatest number.

or

In [28]:

```
1 birth_year = 1990
2 if birth_year < 1980 or birth_year > 1989:
3     print('You were not born in 1980s.')
4 else:
5     print('You were born in 1990s.')
6 print('Done!')
```

You were not born in 1980s.

Done!

not

In [29]:

```
1 birth_year = 1990
2 if not birth_year == 1991:
3     print('The year of birth is not 1991.')
```

The year of birth is not 1991.

In [15]:

```
1 birth_year = int(input('Enter a year of birth: '))
2
3 print(f'The entered year of birth is: {birth_year}')
4
5 if birth_year < 1985 or birth_year == 1991 or birth_year == 1995:
6     print(f'You were born in {birth_year}')
7 else:
8     # For instance, if your year of birth is 1993
9     print(f'Your year of birth with {birth_year} is wrong.')
```

The entered year of birth is: 1993

Your year of birth with 1993 is wrong.

In [16]:

```
1 birth_year = int(input('Enter a year of birth: '))
2
3 print(f'The entered year of birth is: {birth_year}')
4
5 if birth_year < 1985 or birth_year == 1991 or birth_year == 1995:
6     # For instance, if your year of birth is 1995
7     print(f'You were born in {birth_year}')
8 else:
9     print(f'Your year of birth with {birth_year} is wrong.')
```

The entered year of birth is: 1995

You were born in 1995

Python Tutorial

Created by Mustafa Germec, PhD

8. Loops in Python

- A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the **for** keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the **for** loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- The **for** loop does not require an indexing variable to set beforehand.
- With the **while** loop we can execute a set of statements as long as a condition is true.
- Note: remember to increment **i**, or else the loop will continue forever.
- The **while** loop requires relevant variables to be ready, in this example we need to define an indexing variable, **i**, which we set to 1.

range() function

- It is helpful to think of the range object as an ordered list.
- To loop through a set of code a specified number of times, we can use the `range()` function,
- The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

In [3]:

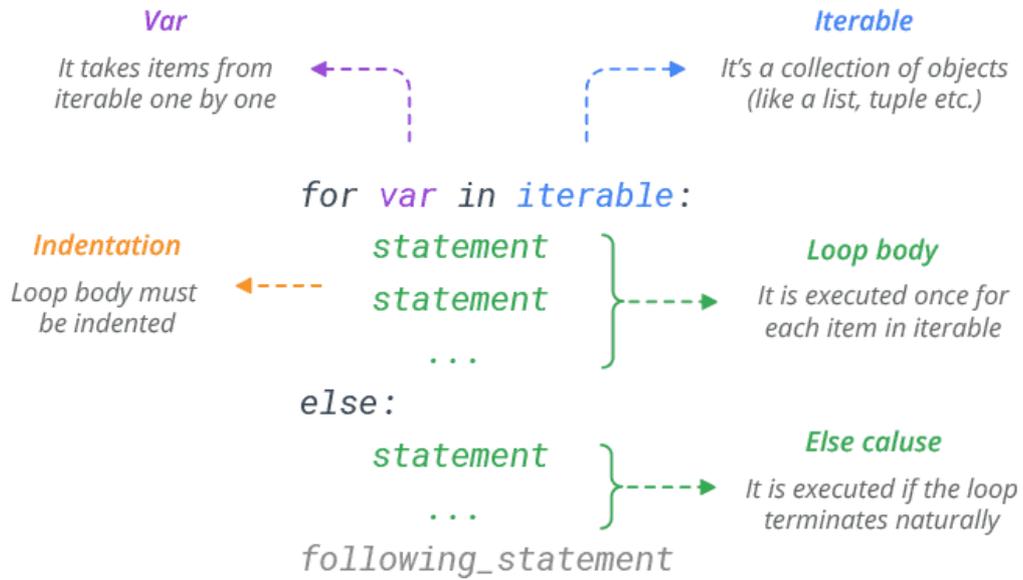
```
1 # Take a range() function
2 print(range(5))
3 print(range(10))
```

```
range(0, 5)
```

```
range(0, 10)
```

for loop

The **for** loop enables you to execute a code block multiple times.



In [4]:

```

1 # Take an example
2 # Directly accessing to the elements in the list
3
4 years = [2005, 2006, 2007, 2008, 2009, 2010]
5
6 for i in years:
7     print(i)

```

```

2005
2006
2007
2008
2009
2010

```

In [10]:

```

1 # Again, directly accessing to the elements in the list
2 years = [2005, 2006, 2007, 2008, 2009, 2010]
3
4 for year in years:
5     print(year)

```

```

2005
2006
2007
2008
2009
2010

```

In [6]:

```
1 # Take an example
2 years = [2005, 2006, 2007, 2008, 2009, 2010]
3
4 for i in range(len(years)):
5     print(years[i])
```

```
2005
2006
2007
2008
2009
2010
```

In [8]:

```
1 # Another for loop example
2 for i in range(2, 12):
3     print(i)
```

```
2
3
4
5
6
7
8
9
10
11
```

In [16]:

```
1 # Striding in for loop
2 for i in range(2, 12, 3):
3     print(i)
```

```
2
5
8
11
```

In [12]:

```
1 # Changing the elements in the list
2 languages = ['Java', 'JavaScript', 'C', 'C++', 'PHP']
3
4 for i in range(len(languages)):
5     print('Before language', i, 'is', languages[i])
6     languages[i] = 'Python'
7     print('After language', i, 'is', languages[i])
```

Before language 0 is Java
After language 0 is Python
Before language 1 is JavaScript
After language 1 is Python
Before language 2 is C
After language 2 is Python
Before language 3 is C++
After language 3 is Python
Before language 4 is PHP
After language 4 is Python

In [14]:

```
1 # Enumaeration of the elements in the list
2 languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']
3
4 for index, language in enumerate(languages):
5     print(index, language)
```

0 Python
1 Java
2 JavaScript
3 C
4 C++
5 PHP

In [30]:

```
1 # Take the numbers between -3 and 6 using for loop
2 # Use range() function
3
4 for i in range(-3, 7):
5     print(i)
```

-3
-2
-1
0
1
2
3
4
5
6

In [31]:

```

1 # Take a list and print the elements using for loop
2 languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']
3
4 for i in range(len(languages)):
5     print(i, languages[i])

```

0 Python

1 Java

2 JavaScript

3 C

4 C++

5 PHP

In [120]:

```

1 number1 = int(input('Enter a number:'))
2 number2 = int(input('Enter a number:'))
3 print(f'The entered numbers are {number1} and {number2}.')
4 for i in range(0, 11):
5     print('%d x %d = %d' %(number1, i, number1*i), ', ', ('%d x %d = %d' %(number2, i, number2*i)))

```

The entered numbers are 7 and 9.

7 x 0 = 0 , 9 x 0 = 0

7 x 1 = 7 , 9 x 1 = 9

7 x 2 = 14 , 9 x 2 = 18

7 x 3 = 21 , 9 x 3 = 27

7 x 4 = 28 , 9 x 4 = 36

7 x 5 = 35 , 9 x 5 = 45

7 x 6 = 42 , 9 x 6 = 54

7 x 7 = 49 , 9 x 7 = 63

7 x 8 = 56 , 9 x 8 = 72

7 x 9 = 63 , 9 x 9 = 81

7 x 10 = 70 , 9 x 10 = 90

Addition and average calculation in for loop

In [2]:

```

1 # Take a list
2 nlis = [0.577, 2.718, 3.14, 1.618, 1729, 6, 37]
3
4 # Write a for loop for addition
5 count = 0
6 for i in nlis:
7     count+=i
8 print('The total value of the numbers in the list is', count)
9
10 # Calculate the average using len() function
11 print('The avearge value of the numbers in the list is', count/len(nlis))

```

The total value of the numbers in the list is 1780.053

The total value of the numbers in the list is 254.29328571428573

for-else statement

In [116]:

```
1 # Take a list
2 nlis = [1,2,4,5,6,7,8,9,10,11,12,13,14]
3 for i in nlis:
4     if i == 5:
5         continue
6     print(i)
7
8 """
9 You see that the output includes the numbers without 5.
10 The continue function jumps when it meets with the reference.
11 """
```

```
1
2
4
6
7
8
9
10
11
12
13
14
```

Out[116]:

'\nYou see that the output includes the numbers without 5. \n\nThe continue function jumps when it meets with the reference.\n'

break in for loop

In [118]:

```
1 # Take a list
2 nlis = [1,2,4,5,6,7,8,9,10,11,12,13,14]
3 for i in nlis:
4     if i == 5:
5         break
6     print(i)
7
8 """
9 You see that the output includes the numbers before 5.
10 The break function terminate the loop when it meets with the reference.
11 """
```

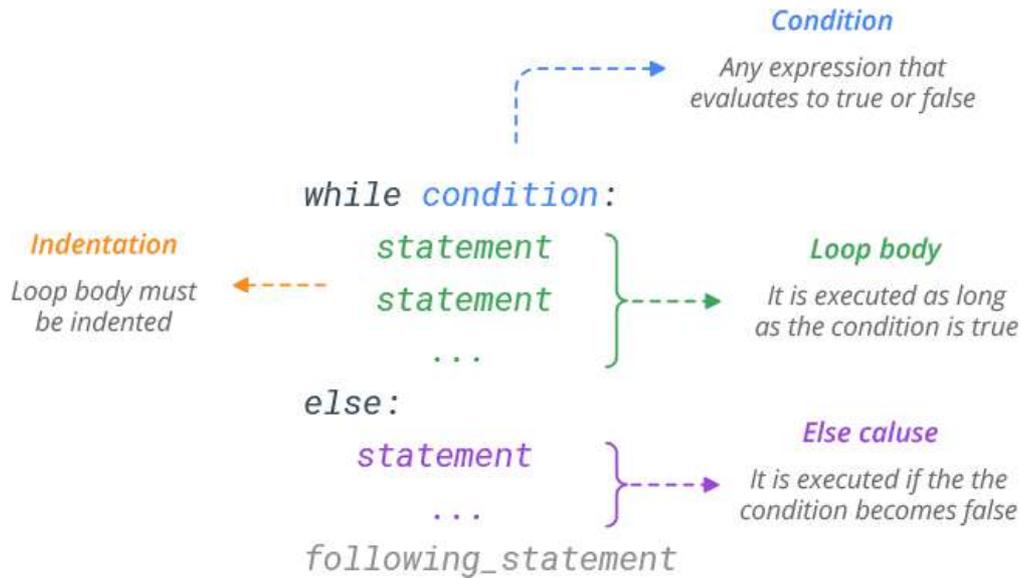
```
1
2
4
```

Out[118]:

'\nYou see that the output includes the numbers before 5. \n\nThe break function terminate the loop when it meets with the reference.\n'

while loop

The **while** loop exists as a tool for repeated execution based on a condition. The code block will keep being executed until the given logical condition returns a *False* boolean value.



In [21]:

```

1 # Take an example
2 i = 22
3 while i < 27:
4     print(i)
5     i += 1
  
```

22
23
24
25
26

In [22]:

```

1 # Take an example
2 i = 22
3 while i >= 17:
4     print(i)
5     i -= 1
  
```

22
21
20
19
18
17

In [25]:

```

1 # Take an example
2 years = [2005, 2006, 2007, 2008, 2009, 2010]
3
4 index = 0
5
6 year = years[0]
7
8 while year !=2008:
9     print(year)
10    index+=1
11    year = years[index]
12 print('It gives us only', index, 'repetitons to get out of loop')
13

```

2005

2006

2007

It gives us only 3 repetitons to get out of loop

In [37]:

```

1 # Print the movie ratings gretater than 6.
2 movie_rating = [8.0, 7.5, 5.4, 9.1, 6.3, 6.5, 2.1, 4.8, 3.3]
3
4 index = 0
5 rating = movie_rating[0]
6
7 while rating>=6:
8     print(rating)
9     index += 1
10    rating = movie_rating[index]
11 print('There is only', index, 'movie rating, because the loop stops when it meets with the number lower than 6.')

```

8.0

7.5

There is only 2 movie rating, because the loop stops when it meets with the number lower than 6.

In [83]:

```

1 # Print the movie ratings gretater than 6.
2 movie_rating = [8.0, 7.5, 5.4, 9.1, 6.3, 6.5, 2.1, 4.8, 3.3]
3
4 index = 0
5 for i in range(len(movie_rating)):
6     if movie_rating[i] >= 6:
7         index += 1
8         print(index, movie_rating[i])
9 print('There is only', index, 'films gretater than movie rating 6')

```

1 8.0

2 7.5

3 9.1

4 6.3

5 6.5

There is only 5 films gretater than movie rating 6

In [91]:

```

1 # Adding the element in a list to a new list
2 fruits = ['banana', 'apple', 'banana', 'orange', 'kiwi', 'banana', 'Cherry', 'Grapes']
3
4 new_fruits = []
5
6 index = 0
7 while fruits[index] == 'banana':
8     new_fruits.append(fruits[index])
9     index += 1
10 print(new_fruits)

```

['banana']

In [119]:

```

1 number1 = int(input('Enter a number:'))
2 number2 = int(input('Enter a number:'))
3 print(f'The entered numbers are {number1} and {number2}.')
4
5 i = 0
6 while i<=10:
7     print('%d x %d = %d' %(number1, i, number1*i)), ', ', ('%d x %d = %d' %(number2, i, number2*i)))
8     i+=1
9

```

The entered numbers are 8 and 9.

```

8 x 0 = 0 , 9 x 0 = 0
8 x 1 = 8 , 9 x 1 = 9
8 x 2 = 16 , 9 x 2 = 18
8 x 3 = 24 , 9 x 3 = 27
8 x 4 = 32 , 9 x 4 = 36
8 x 5 = 40 , 9 x 5 = 45
8 x 6 = 48 , 9 x 6 = 54
8 x 7 = 56 , 9 x 7 = 63
8 x 8 = 64 , 9 x 8 = 72
8 x 9 = 72 , 9 x 9 = 81
8 x 10 = 80 , 9 x 10 = 90

```

while-else statement

In [29]:

```

1 index = 0
2 while index <=5:
3     print(index, end=' ')
4     index += 1
5 else:
6     print('It gives us the numbers between 0 and 5.')

```

0 1 2 3 4 5 It gives us the numbers between 0 and 5.

continue in while loop

In [122]:

```
1 i = 0
2
3 while i<=5:
4     print(i)
5     i+=1
6     if i == 3:
7         continue
```

```
0
1
2
3
4
5
```

break in while loop

In [121]:

```
1 i = 0
2
3 while i<=5:
4     print(i)
5     i+=1
6     if i == 3:
7         break
```

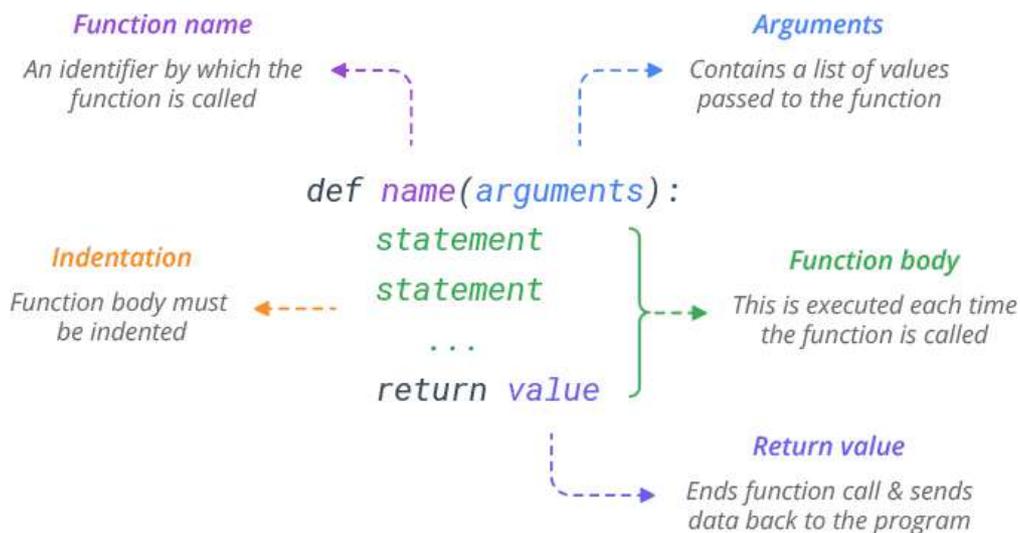
```
0
1
2
```

Python Tutorial

Created by Mustafa Germec, PhD

9. Functions in Python

- In Python, a **function** is a group of related statements that performs a specific task.
- **Functions** help break our program into smaller and modular chunks. * As our program grows larger and larger, **functions** make it more organized and manageable.
- Furthermore, it avoids repetition and makes the code reusable.
- There are two types of functions :
- **Pre-defined functions**
- **User defined functions**
- In Python a **function** is defined using the **def** keyword followed by the function **name** and parentheses (**()**).
- Keyword **def** that marks the start of the function header.
- A **function** name to uniquely identify the **function**.
- **Function** naming follows the same rules of writing identifiers in Python.
- Parameters (arguments) through which we pass values to a **function**. They are optional.
- A **colon (:)** to mark the end of the **function** header.
- Optional documentation string (docstring) to describe what the **function** does.
- One or more valid python statements that make up the **function** body.
- Statements must have the same indentation level (usually 4 spaces).
- An optional return statement to return a value from the **function**.



In [9]:

```
1 # Take a function sample
2 # Mathematical operations in a function
3
4 def process(x):
5     y1 = x-8
6     y2 = x+8
7     y3 = x*8
8     y4 = x/8
9     y5 = x%8
10    y6 = x//8
11    print(f'If you make the above operations with {x}, the results will be {y1}, {y2}, {y3}, {y4}, {y5}, {y6}.')
12    return y1, y2, y3, y4, y5, y6
13
14 process(5)
```

If you make the above operations with 5, the results will be -3, 13, 40, 0.625, 5, 0.

Out[9]:

```
(-3, 13, 40, 0.625, 5, 0)
```

You can request help using help() function

In [10]:

```
1 help(process)
```

Help on function process in module __main__:

```
process(x)
```

Call the function again with the number 3.14

In [11]:

```
1 process(3.14)
```

If you make the above operations with 3.14, the results will be -4.859999999999999, 11.14, 25.12, 0.3925, 3.14, 0.0.

Out[11]:

```
(-4.859999999999999, 11.14, 25.12, 0.3925, 3.14, 0.0)
```

Functions with multiple parameters

In [2]:

```

1 # Define a function with multiple elements
2 def mult(x, y):
3     z = 2*x + 5*y + 45
4     return z
5
6 output = mult(3.14, 1.618) # You can yield the output by assigning to a variable
7 print(output)
8 print(mult(3.14, 1.618)) # You can obtain the result directly
9 mult(3.14, 1.618) # This is also another version

```

59.370000000000005

59.370000000000005

Out[2]:

59.370000000000005

In [20]:

```

1 # Call again the defined function with different arguments
2 print(mult(25, 34))

```

265

Variables

- The input to a function is called a **formal parameter**.
- A **variable** that is declared inside a function is called a **local variable**.
- The parameter only exists within the function (i.e. the point where the function starts and stops).
- A **variable** that is declared outside a function definition is a **global variable**, and its value is accessible and modifiable throughout the program.

In [5]:

```

1 # Define a function
2 def function(x):
3
4     # Take a local variable
5     y = 3.14
6     z = 3*x + 1.618*y
7     print(f'If you make the above operations with {x}, the results will be {z}.')
8     return z
9
10 with_golden_ratio = function(1.618)
11 print(with_golden_ratio)

```

If you make the above operations with 1.618, the results will be 9.934520000000001.

9.934520000000001

In [8]:

```
1 # It starts the global variable
2 a = 3.14
3
4 # call function and return function
5 y = function(a)
6 print(y)
```

If you make the above operations with 3.14, the results will be 14.500520000000002.
14.500520000000002

In [9]:

```
1 # Enter a number directly as a parameter
2 function(2.718)
```

If you make the above operations with 2.718, the results will be 13.23452.

Out[9]:

13.23452

Without return statement, the function returns None

In [10]:

```
1 # Define a function with and without return statement
2 def msg1():
3     print('Hello, Python!')
4
5 def msg2():
6     print('Hello, World!')
7     return None
8
9 msg1()
10 msg2()
```

Hello, Python!

Hello, World!

In [15]:

```
1 # Printing the function after a call indicates a None is the default return statement.
2 # See the following printings what functions returns are.
3
4 print(msg1())
5 print(msg2())
```

Hello, Python!

None

Hello, World!

None

Concatenation of two strings

In [18]:

```
1 # Define a function
2 def strings(x, y):
3     return x + y
4
5 # Testing the function 'strings(x, y)'
6 strings('Hello', 'Python')
```

Out[18]:

'Hello Python'

Simplicity of functions

In [26]:

```
1 # The following codes are not used again.
2 x = 2.718
3 y = 0.577
4 equation = x*y + x+y - 37
5 if equation>0:
6     equation = 6
7 else: equation = 37
8
9 equation
```

Out[26]:

37

In [27]:

```
1 # The following codes are not used again.
2 x = 0
3 y = 0
4 equation = x*y + x+y - 37
5 if equation<0:
6     equation = 0
7 else: equation = 37
8
9 equation
```

Out[27]:

0

In [28]:

```
1 # The following codes can be write as a function.
2 def function(x, y):
3     equation = x*y + x+y - 37
4     if equation>0:
5         equation = 6
6     else: equation = 37
7     return equation
8
9 x = 2.718
10 y = 0.577
11 function(x, y)
```

Out[28]:

37

In [29]:

```
1 # The following codes can be write as a function.
2 def function(x, y):
3     equation = x*y + x+y - 37
4     if equation<0:
5         equation = 6
6     else: equation = 37
7     return equation
8
9 x = 0
10 y = 0
11 function(x, y)
```

Out[29]:

6

Predefined functions like print(), sum(), len(), min(), max(), input()

In [31]:

```
1 # print() is a built-in function
2 special_numbers = [0.577, 2.718, 3.14, 1.618, 1729, 6, 28, 37]
3 print(special_numbers)
```

[0.577, 2.718, 3.14, 1.618, 1729, 6, 28, 37]

In [32]:

```
1 # The function sum() add all elements in a list or a tuple
2 sum(special_numbers)
```

Out[32]:

1808.053

In [33]:

```

1 # The function len() gives us the length of the list or tuple
2 len(special_numbers)

```

Out[33]:

8

Using conditions and loops in functions

In [44]:

```

1 # Define a function including conditions if/else
2
3 def fermentation(microorganism, substrate, product, activity):
4     print(microorganism, substrate, product, activity)
5     if activity < 1000:
6         return f'The fermentation process was unsuccessful with the {product} activity of {activity} U/mL from {substrate}'
7     else:
8         return f'The fermentation process was successful with the {product} activity of {activity} U/mL from {substrate} using {microorganism}'
9
10 result1 = fermentation('Aspergillus niger', 'molasses', 'inulinase', 1800)
11 print(result1)
12 print()
13 result2 = fermentation('Aspergillus niger', 'molasses', 'inulinase', 785)
14 print(result2)
15

```

Aspergillus niger molasses inulinase 1800

The fermentation process was successful with the inulinase activity of 1800 U/mL from molasses using Aspergillus niger.

Aspergillus niger molasses inulinase 785

The fermentation process was unsuccessful with the inulinase activity of 785 U/mL from molasses using Aspergillus niger. You should repeat the fermentation process.

In [50]:

```

1 # Define a function using the loop 'for'
2
3 def fermentation(content):
4     for parameters in content:
5         print(parameters)
6
7 content = ['Stirred-tank bioreactor', '30°C temperature', '200 rpm agitation speed', '1 vvm aeration', '1% (v/v) inoculum ratio', 'pH control at 5.0']
8 fermentation(content)

```

Stirred-tank bioreactor

30°C temperature

200 rpm agitation speed

1 vvm aeration

1% (v/v) inoculum ratio

pH control at 5.0

Adjusting default values of independent variables in functions

In [53]:

```

1 # Define a function adjusting the default value of the variable
2
3 def rating_value(rating = 5.5):
4     if rating < 8:
5         return f'You should not watch this film with the rating value of {rating}'
6     else:
7         return f'You should watch this film with the rating value of {rating}'
8
9 print(rating_value())
10 print(rating_value(8.6))

```

You should not watch this film with the rating value of 5.5

You should watch this film with the rating value of 8.6

Global variables

- Variables that are created outside of a function (as in all of the examples above) are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

In [56]:

```

1 # Define a function for a global variable
2 language = 'Python'
3
4 def lang(language):
5     global_var = language
6     print(f'{language} is a program language.')
7
8 lang(language)
9 lang(global_var)
10
11 """
12 The output gives a NameError, since all variables in the function are local variables,
13 so variable assignment is not persistent outside the function.
14 """

```

Python is a program language.

```

-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_21468\4270999454.py in <module>
7
8 lang(language)
----> 9 lang(global_var)

```

NameError: name 'global_var' is not defined

In [58]:

```
1 # Define a function for a global variable
2 language = 'JavaScript'
3
4 def lang(language):
5     global global_var
6     global_var = 'Python'
7     print(f'{language} is a programing language.')
8
9 lang(language)
10 lang(global_var)
```

JavaScript is a programing language.

Python is a programing language.

Variables in functions

- The scope of a variable is the part of the program to which that variable is accessible.
- Variables declared outside of all function definitions can be accessed from anywhere in the program.
- Consequently, such variables are said to have global scope and are known as global variables.

In [76]:

```
1 process = 'Continuous fermentation'
2
3 def fermentation(process_name):
4     if process_name == process:
5         return '0.5 g/L/h.'
6     else:
7         return '0.25 g/L/h.'
8
9 print('The productivity in continuous fermentation is', fermentation('Continuous fermentation'))
10 print('The productivity in batch fermentation is', fermentation('Batch fermentation'))
11 print('Continuous fermentation has many advantages over batch fermentation.')
12 print(f'My favourite process is {process}.')
```

The productivity in continuous fermentation is 0.5 g/L/h.

The productivity in batch fermentation is 0.25 g/L/h.

Continuous fermentation has many advantages over batch fermentation.

My favourite process is Continuous fermentation.

In [77]:

```

1 # If the variable 'process' is deleted, it returns a NameError as follows
2 del process
3
4 # Since the variable 'process' is deleted, the following function is an example of local variable
5 def fermentation(process_name):
6     process = 'Continuous fermentation'
7     if process_name == process:
8         return '0.5 g/L/h.'
9     else:
10        return '0.25 g/L/h.'
11
12 print('The productivity in continuous fermentation is', fermentation('Continuous fermentation'))
13 print('The productivity in batch fermentation is', fermentation('Batch fermentation'))
14 print('Continuous fermentation has many advantages over batch fermentation.')
15 print(f'My favourite process is {process}.')

```

The productivity in continuous fermentation is 0.5 g/L/h.

The productivity in batch fermentation is 0.25 g/L/h.

Continuous fermentation has many advantages over batch fermentation.

```

-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_21468\2006816728.py in <module>
    13 print('The productivity in batch fermentation is', fermentation('Batch fermentation'))
    14 print('Continuous fermentation has many advantages over batch fermentation.')
--> 15 print(f'My favourite process is {process}.')

```

NameError: name 'process' is not defined

In [81]:

```

1 # When the global variable and local variable have the same name:
2
3 process = 'Continuous fermentation'
4
5 def fermentation(process_name):
6     process = 'Batch fermentation'
7     if process_name == process:
8         return '0.5 g/L/h.'
9     else:
10        return '0.25 g/L/h.'
11
12 print('The productivity in continuous fermentation is', fermentation('Continuous fermentation'))
13 print('The productivity in batch fermentation is', fermentation('Batch fermentation'))
14 print(f'My favourite process is {process}.')

```

The productivity in continuous fermentation is 0.25 g/L/h.

The productivity in batch fermentation is 0.5 g/L/h.

My favourite process is Continuous fermentation.

(args) and/or (*args) and Functions

When the number of arguments are unknown for a function, then the arguments can be packet into a tuple or a dictionary

In [84]:

```
1 # Define a function regarding a tuple example
2 def function(*args):
3     print('Number of elements is', len(args))
4     for element in args:
5         print(element)
6
7 function('Aspergillus niger', 'inulinase', 'batch', '1800 U/mL activity')
8 print()
9 function('Saccharomyces cerevisia', 'ethanol', 'continuous', '45% yield', 'carob')
10
```

Number of elements is 4
Aspergillus niger
inulinase
batch
1800 U/mL activity

Number of elements is 5
Saccharomyces cerevisia
ethanol
continuous
45% yield
carob

In [98]:

```
1 # Another example regarding 'args'
2 def total(*args):
3     total = 0
4     for i in args:
5         total += i
6     return total
7
8 print('The total of the numbers is', total(0.577, 2.718, 3.14, 1.618, 1729, 6, 37))
```

The total of the numbers is 1780.053

In [88]:

```
1 # Define a function regarding a dictionary example
2 def function(**args):
3     for key in args:
4         print(key, ':', args[key])
5
6 function(Micoorganism='Aspergillus niger', Substrate='Molasses', Product='Inulinase', Fermentation_mode='Batch', A
```

Micoorganism : Aspergillus niger
Substrate : Molasses
Product : Inulinase
Fermentation_mode : Batch
Activity : 1800 U/mL

In [96]:

```
1 # Define a function regarding the addition of elements into a list
2 def addition(nlist):
3     nlist.append(3.14)
4     nlist.append(1.618)
5     nlist.append(1729)
6     nlist.append(6)
7     nlist.append(37)
8
9 my_list= [0.577, 2.718]
10 addition(my_list)
11 print(my_list)
12 print(sum(my_list))
13 print(min(my_list))
14 print(max(my_list))
15 print(len(my_list))
```

```
[0.577, 2.718, 3.14, 1.618, 1729, 6, 37]
1780.053
0.577
1729
7
```

Doctsting in Functions

In [97]:

```
1 # Define a function
2 def addition(x, y):
3     """The following function returns the sum of two parameters."""
4     z = x+y
5     return z
6
7 print(addition.__doc__)
8 print(addition(3.14, 2.718))
```

The following function returns the sum of two parameters.
5.8580000000000005

Recursive functions

In [103]:

```
1 # Calculating the factorial of a certain number.
2
3 def factorial(number):
4     if number == 0:
5         return 1
6     else:
7         return number*factorial(number-1)
8
9 print('The value is', factorial(6))
```

The value is 720

In [107]:

```
1 # Define a function that gives the total of the first ten numbers
2 def total_numbers(number, sum):
3     if number == 11:
4         return sum
5     else:
6         return total_numbers(number+1, sum+number)
7
8 print('The total of first ten numbers is', total_numbers(1, 0))
```

The total of first ten numbers is 55

Nested functions

In [111]:

```
1 # Define a function that add a number to another number
2 def added_num(num1):
3     def incremented_num(num1):
4         num1 = num1 + 1
5         return num1
6     num2 = incremented_num(num1)
7     print(num1, '----->>', num2)
8
9 added_num(25)
```

25 ----->> 26

nonlocal function

In [112]:

```
1 # Define a function regarding 'nonlocal' function
2 def print_year():
3     year = 1990
4     def print_current_year():
5         nonlocal year
6         year += 32
7         print('Current year is', year)
8     print_current_year()
9 print_year()
```

Current year is 2022

In [117]:

```
1 # Define a function giving a message
2 def function(name):
3     msg = 'Hi ' + name
4     return msg
5
6 name = input('Enter a name: ')
7 print(function(name))
```

Hi Mustafa

Python Tutorial

Created by Mustafa Germec, PhD

10. Exception Handling in Python

- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.
- An exception is a Python object that represents an error.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.
- If you have some **suspicious** code that may raise an exception, you can defend your program by placing the **suspicious** code in a **try:** block.
- After the **try:** block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.
- **Common exceptions**
 - ZeroDivisionError
 - NameError
 - ValueError
 - IOError
 - EOFError
 - IndentationError

ZeroDivisionError

In [1]:

```
1 # If a number is divided by 0, it gives a ZeroDivisionError.
2 try:
3     1/0
4 except ZeroDivisionError:
5     print('This code gives a ZeroDivisionError.')
6
7 print(1/0)
```

This code gives a ZeroDivisionError.

```
-----
ZeroDivisionError      Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\3605061481.py in <module>
      5 print('This code gives a ZeroDivisionError.')
      6
----> 7 print(1/0)
```

ZeroDivisionError: division by zero

In [2]:

```
1 nlis = []
2 count = 0
3 try:
4     mean = count/len(nlis)
5     print('The mean value is', mean)
6 except ZeroDivisionError:
7     print('This code gives a ZeroDivisionError')
8
9 print(count/len(nlis))
```

This code gives a ZeroDivisionError

```
-----
ZeroDivisionError      Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\2225123637.py in <module>
      7 print('This code gives a ZeroDivisionError')
      8
----> 9 print(count/len(nlis))
```

ZeroDivisionError: division by zero

In [3]:

```
1 # The following code is like 1/0.
2 try:
3     True/False
4 except ZeroDivisionError:
5     print('The code gives a ZeroDivisionError.!)
6
7 print(True/False)
```

The code gives a ZeroDivisionError.

```
-----
ZeroDivisionError      Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\3531407864.py in <module>
      5 print('The code gives a ZeroDivisionError.!)
      6
----> 7 print(True/False)
```

ZeroDivisionError: division by zero

NameError

In [4]:

```
1 nlis = []
2 count = 0
3 try:
4     mean = count/len(nlis)
5     print('The mean value is', mean)
6 except ZeroDivisionError:
7     print('This code gives a ZeroDivisionError')
8
9 # Since the variable 'mean' is not defined, it gives us a 'NameError
10 print(mean)
```

This code gives a ZeroDivisionError

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\1642249892.py in <module>
      8
      9 # Since the variable 'mean' is not defined, it gives us a 'NameError
----> 10 print(mean)
```

NameError: name 'mean' is not defined

In [5]:

```
1 try:
2     y = x+5
3 except NameError:
4     print('This code gives a NameError.')
5
6 print(y)
```

This code gives a NameError.

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\115043188.py in <module>
      4 print('This code gives a NameError.')
      5
----> 6 print(y)
```

NameError: name 'y' is not defined

In [6]:

```
1 # Define a function giving a NameError
2 def addition(x, y):
3     z = x + y
4     return z
5
6 print('This function gives a NameError.')
7 total = add(3.14, 1.618)
8 print(total)
```

This function gives a NameError.

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\3845321401.py in <module>
      5
      6 print('This function gives a NameError.')
----> 7 total = add(3.14, 1.618)
      8 print(total)
```

NameError: name 'add' is not defined

In [7]:

```
1 # Since 'Mustafa' is not defined, the following code gives us a 'NameError.'
2 try:
3     name = (Mustafa)
4     print(name, 'today is your wedding day.')
5 except NameError:
6     print('This code gives a NameError.')
7
8 name = (Mustafa)
9 print(name, 'today is your wedding day.')
```

This code gives a NameError.

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\367854978.py in <module>
      6 print('This code gives a NameError.')
      7
----> 8 name = (Mustafa)
      9 print(name, 'today is your wedding day.')
```

NameError: name 'Mustafa' is not defined

IndexError

In [8]:

```
1 nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 try:
3     nlis[10]
4 except IndexError:
5     print('This code gives us a IndexError.')
6
7 print(nlis[10])
```

This code gives us a IndexError.

```
-----
IndexError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\4262347625.py in <module>
      5 print('This code gives us a IndexError.')
      6
----> 7 print(nlis[10])
```

IndexError: list index out of range

In [9]:

```
1 # You can also supplytake this error type with tuple
2 tuple_sample = (0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729)
3 try:
4     tuple_sample[10]
5 except IndexError:
6     print('This code gives us a IndexError.')
7
8 print(tuple_sample[10])
```

This code gives us a IndexError.

```
-----
IndexError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\3170854299.py in <module>
      6 print('This code gives us a IndexError.')
      7
----> 8 print(tuple_sample[10])
```

IndexError: tuple index out of range

KeyError

In [10]:

```

1 dictionary = {'euler_constant': 0.577, 'golden_ratio': 1.618}
2 try:
3     dictionary = dictionary['euler_number']
4 except KeyError:
5     print('This code gives us a KeyError.')
6
7 dictionary = dictionary['euler_number']
8 print(dictionary)

```

This code gives us a KeyError.

```

-----
KeyError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5432\669363184.py in <module>
      5 print('This code gives us a KeyError.')
      6
----> 7 dictionary = dictionary['euler_number']
      8 print(dictionary)

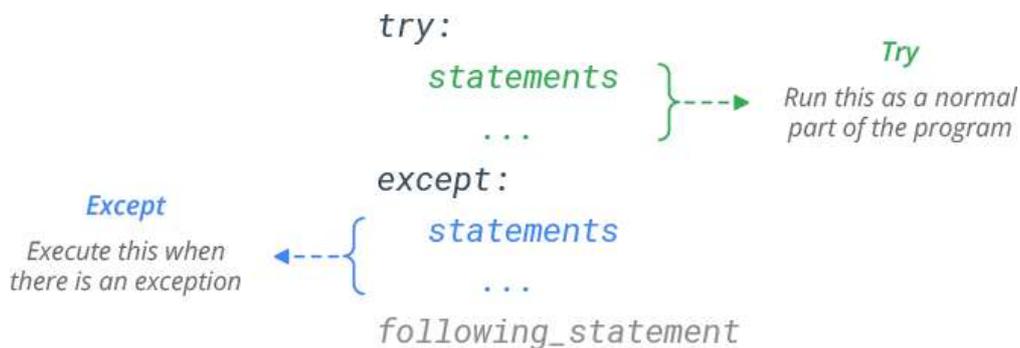
```

KeyError: 'euler_number'

You can find more [Error Types \(https://docs.python.org/3/library/exceptions.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=N_SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01\)](https://docs.python.org/3/library/exceptions.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_term=10006555&utm_id=N_SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01) from this connection.

Exception Handling

try/except



In [11]:

```

1 try:
2     print(name)
3 except NameError:
4     print('Since the variable name is not defined, the function gives a NameError.')

```

Since the variable name is not defined, the function gives a NameError.

In [1]:

```

1 num1 = float(input('Enter a number:'))
2 print('The entered value is', num1)
3 try:
4     num2 = float(input('Enter a number:'))
5     print('The entered value is', num2)
6     value = num1/num2
7     print('This process is running with value = ', value)
8 except:
9     print('This process is not running.')

```

The entered value is 3.14

The entered value is 0.577

This process is running with value = 5.441941074523397

Multiple Except Blocks

try/except/except etc.

In [2]:

```

1 num1 = float(input('Enter a number:'))
2 print('The entered value is', num1)
3 try:
4     num2 = float(input('Enter a number:'))
5     print('The entered value is', num2)
6     value = num1/num2
7     print('This process is running with value = ', value)
8 except ZeroDivisionError:
9     print('This function gives a ZeroDivisionError since a number cannot divide by 0.')
10 except ValueError:
11     print('You should provide a number.')
12 except:
13     print('Soething went wrong!')

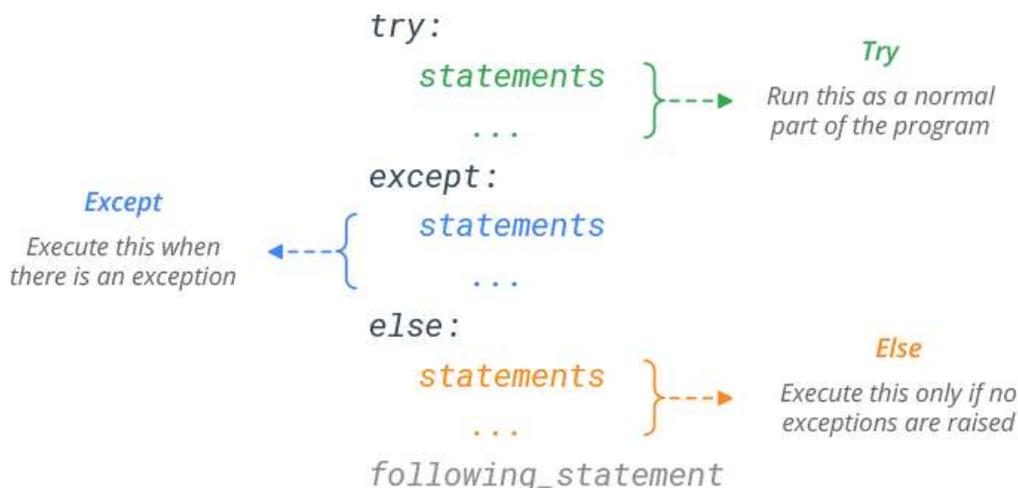
```

The entered value is 2.718

The entered value is 0.0

This function gives a ZeroDivisionError since a number cannot divide by 0.

try/except/else



In [3]:

```

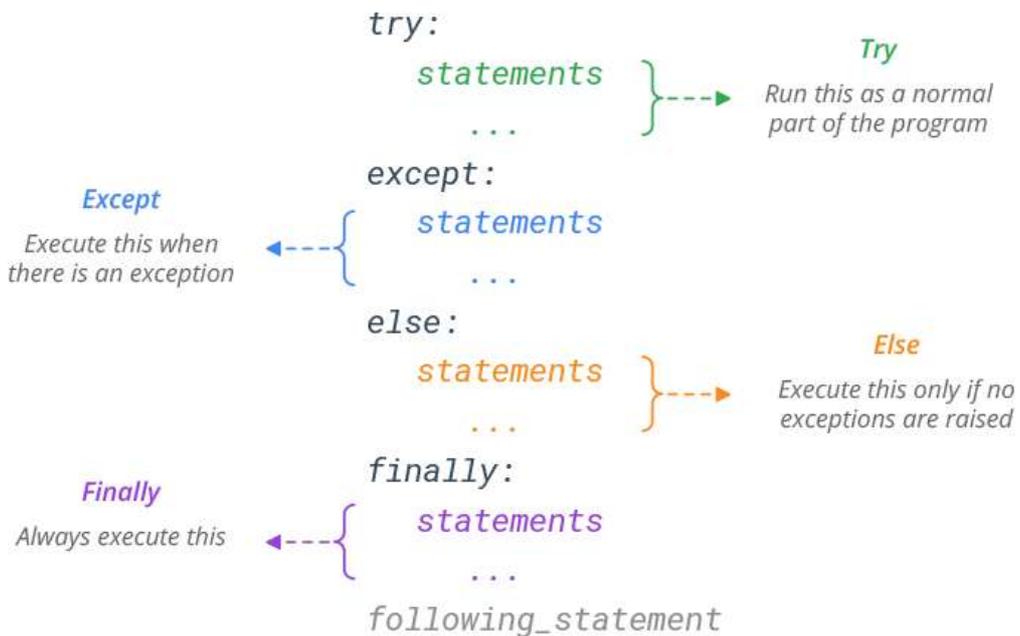
1 num1 = float(input('Enter a number:'))
2 print('The entered value is', num1)
3 try:
4     num2 = float(input('Enter a number:'))
5     print('The entered value is', num2)
6     value = num1/num2
7 except ZeroDivisionError:
8     print('This function gives a ZeroDivisionError since a number cannot divide by 0.')
9 except ValueError:
10    print('You should provide a number.')
11 except:
12    print('Soething went wrong!')
13 else:
14    print('This process is running with value = ', value)

```

The entered value is 37.0

The entered value is 1.618

This process is running with value = 22.867737948084052

try/except/else/finally

In [5]:

```
1 num1 = float(input('Enter a number:'))
2 print('The entered value is', num1)
3 try:
4     num2 = float(input('Enter a number:'))
5     print('The entered value is', num2)
6     value = num1/num2
7 except ZeroDivisionError:
8     print('This function gives a ZeroDivisionError since a number cannot divide by 0.')
9 except ValueError:
10    print('You should provide a number.')
11 except:
12    print('Soething went wrong!')
13 else:
14    print('This process is running with value = ', value)
15 finally:
16    print('The process is completed.')
```

The entered value is 1.618

The entered value is 0.577

This process is running with value = 2.8041594454072793

The process is completed.

Multiple except clauses

In [6]:

```
1 num1 = float(input('Enter a number:'))
2 print('The entered value is', num1)
3 try:
4     num2 = float(input('Enter a number:'))
5     print('The entered value is', num2)
6     value = num1/num2
7 except (ZeroDivisionError, NameError, ValueError): #Multiple except clauses
8     print('This function gives a ZeroDivisionError, NameError or ValueError.')
9 except:
10    print('Soething went wrong!')
11 else:
12    print('This process is running with value = ', value)
13 finally:
14    print('The process is completed.')
```

The entered value is 3.14

The entered value is 0.0

This function gives a ZeroDivisionError, NameError or ValueError.

The process is completed.

Raising in exception

Using the 'raise' keyword, the programmer can throw an exception when a certain condition is reached.

In [7]:

```
1 num = int(input('Enter a number:'))
2 print('The entered value is', num)
3 try:
4     if num>1000 and num %2 == 0 or num %2 !=0:
5         raise Exception('Do not allow to the even numbers higher than 1000.')
6 except:
7     print('Even or odd numbers higher than 1000 are not allowed!')
8 else:
9     print('This process is running with value = ', num)
10 finally:
11     print('The process is completed.')
```

The entered value is 1006

Even or odd numbers higher than 1000 are not allowed!

The process is completed.

Python Tutorial

Created by Mustafa Germec, PhD

11. Built-in Functions in Python

Python has several functions that are readily available for use. These functions are called built-in functions. You can find more information about built-in functions from this [Link](https://www.w3schools.com/python/python_ref_functions.asp). (https://www.w3schools.com/python/python_ref_functions.asp).

Built-in Functions			
A <code>abs()</code> <code>all()</code> <code>any()</code> <code>ascii()</code>	E <code>enumerate()</code> <code>eval()</code> <code>exec()</code>	L <code>len()</code> <code>list()</code> <code>locals()</code>	R <code>range()</code> <code>repr()</code> <code>reversed()</code> <code>round()</code>
B <code>bin()</code> <code>bool()</code> <code>breakpoint()</code> <code>bytearray()</code> <code>bytes()</code>	F <code>filter()</code> <code>float()</code> <code>format()</code> <code>frozenset()</code>	M <code>map()</code> <code>max()</code> <code>memoryview()</code> <code>min()</code>	S <code>set()</code> <code>setattr()</code> <code>slice()</code> <code>sorted()</code> <code>staticmethod()</code> <code>str()</code> <code>sum()</code> <code>super()</code>
C <code>callable()</code> <code>chr()</code> <code>classmethod()</code> <code>compile()</code> <code>complex()</code>	G <code>getattr()</code> <code>globals()</code>	N <code>next()</code>	T <code>tuple()</code> <code>type()</code>
D <code>delattr()</code> <code>dict()</code> <code>dir()</code> <code>divmod()</code>	H <code>hasattr()</code> <code>hash()</code> <code>help()</code> <code>hex()</code>	O <code>object()</code> <code>oct()</code> <code>open()</code> <code>ord()</code>	V <code>vars()</code>
	I <code>id()</code> <code>input()</code> <code>int()</code> <code>isinstance()</code> <code>issubclass()</code> <code>iter()</code>	P <code>pow()</code> <code>print()</code> <code>property()</code>	Z <code>zip()</code>
			_ <code>__import__()</code>

abs()

Returns the absolute value of a number

In [3]:

```
1 num1 = int(input('Enter a number: '))
2 print('The entered number is', num1)
3 num2 = float(input('Enter a number: '))
4 print('The entered number is', num2)
5 print('The absolute value of the first number is', abs(num1))
6 print('The absolute number of the second number is', abs(num2))
7 print('The difference between the two numbers is', abs(num1-num2))
```

The entered number is -6

The entered number is -37.0

The absolute value of the first number is 6

The absolute number of the second number is 37.0

The difference between the two numbers is 31.0

all()

Returns *True* if all elements in passes iterable are true. When the iterable object is empty, it returns *True*. Here, 0 and False return *False* in this function.

In [10]:

```
1 nlis1 = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 print(all(nlis1))
3 nlis1.append(0) # Add '0' to the end of the list
4 print(nlis1)
5 print(all(nlis1))
6 nlis1.append(False) # Adds 'False' to the end of the list
7 print(nlis1)
8 print(all(nlis1))
9 nlis1.clear() # It returns an empty list
10 print(nlis1)
11 print(all(nlis1))
```

True

[0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729, 0]

False

[0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729, 0, False]

False

[]

True

bin()

Returns the binary representation of a specified integer

In [14]:

```
1 num = int(input('Enter a number: '))
2 print(f'The entered number is {num}.')
3 print(f'The binary representation of {num} is {bin(num)}.')
```

The entered number is 37.

The binary representation of 37 is 0b100101.

bool()

Converts a value to *boolean*, namely *True* and *False*

In [25]:

```
1 lis, dict, tuple = [], {}, ()
2 print(bool(lis), bool(dict), bool(tuple))
3 lis, dict, tuple = [0], {'a': 1}, (1,)
4 print(bool(lis), bool(dict), bool(tuple))
5 lis, dict, tuple = [0.0], {'a': 1.0}, (1.0,)
6 print(bool(lis), bool(dict), bool(tuple))
7 a, b, c = 0, 3.14, 'Hello, Python!'
8 print(bool(a), bool(b), bool(c))
9 statement = None
10 print(bool(None))
11 true = True
12 print(bool(true))
```

```
False False False
True True True
True True True
False True True
False
True
```

bytes()

Returns a *bbytes* object

In [26]:

```
1 msg = 'Hello, Python!'
2 new_msg = bytes(msg, 'utf-8')
3 print(new_msg)
```

```
b'Hello, Python!'
```

callable()

Checks and returns *True* if the object passed appears to be *callable*

In [31]:

```
1 var = 3.14
2 print(callable(var)) # since the object does not appear callable, it returns False
3
4 def function(): # since the object appears callable, it returns True
5     print('Hi, Python!')
6 msg = function
7 print(callable(msg))
```

```
False
True
```

chr()

It returns a character from the specified Unicode code.

In [134]:

```
1 print(chr(66))
2 print(chr(89))
3 print(chr(132))
4 print(chr(1500))
5 print(chr(3))
6 print(chr(-500))    # The argument must be inside of the range.
```

B
Y

↳

```
-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_16192\1238010354.py in <module>
      4 print(chr(1500))
      5 print(chr(3))
----> 6 print(chr(-500))    # The argument must be inside of the range.
```

ValueError: chr() arg not in range(0x110000)

In [135]:

```
1 print(chr('Python'))    # The argument must be integer.
```

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_16192\213842990.py in <module>
----> 1 print(chr('Python'))    # The argument must be integer.
```

TypeError: 'str' object cannot be interpreted as an integer

compile()

Returns a code object that can subsequently be executed by `exec()` function

In [35]:

```
1 code_line = 'x=3.14\ny=2.718\nprint("Result =", 2*x+5*y)'
2 code = compile(code_line, 'Result.py', 'exec')
3 print(type(code))
4 exec(code)
```

<class 'code'>
Result = 19.87

exec()

Executes the specified code or object

In [39]:

```
1 var = 3.14
2 exec('print(var==3.14)')
3 exec('print(var!=3.14)')
4 exec('print(var+2.718)')
```

True

False

5.8580000000000005

getattr()

It returns the value of the specified attribute (property or method). If it is not found, it returns the default value.

In [42]:

```
1 class SpecialNumbers:
2     euler_constant = 0.577
3     euler_number = 2.718
4     pi = 3.14
5     golden_ratio = 1.618
6     msg = 'These numbers are special.'
7
8 special_numbers = SpecialNumbers()
9 print('The euler number is', getattr(special_numbers, 'euler_number'))
10 print('The golden ratio is', special_numbers.golden_ratio)
```

The euler number is 2.718

The golden ratio is 1.618

delattr()

It deletes the specified attribute (property or method) from the specified object.

In [143]:

```

1 class SpecialNumbers:
2     euler_constant = 0.577
3     euler_number = 2.718
4     pi = 3.14
5     golden_ratio = 1.618
6     msg = 'These numbers are special.'
7
8     def parameter(self):
9         print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)
10
11 special_numbers = SpecialNumbers()
12 special_numbers.parameter()
13 delattr(SpecialNumbers, 'msg') # The code deleted the 'msg'.
14 special_numbers.parameter() # Since the code deleted the 'msg', it returns an AttributeError.

```

0.577 2.718 3.14 1.618 These numbers are special.

```

-----
AttributeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_16192\3892590851.py in <module>
    12 special_numbers.parameter()
    13 delattr(SpecialNumbers, 'msg')
--> 14 special_numbers.parameter()

~\AppData\Local\Temp\ipykernel_16192\3892590851.py in parameter(self)
     7
     8     def parameter(self):
--> 9         print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)
    10
    11 special_numbers = SpecialNumbers()

```

AttributeError: 'SpecialNumbers' object has no attribute 'msg'

dict()

It returns a dictionary (Array).

In [158]:

```

1 name = dict()
2 print(name)
3
4 dictionary = dict(euler_constant = 0.577, euler_number=2.718, golden_ratio=1.618)
5 print(dictionary)

```

```

{}
{'euler_constant': 0.577, 'euler_number': 2.718, 'golden_ratio': 1.618}

```

enumerate()

It takes a collection (e.g. a tuple) and returns it as an enumerate object.

In [156]:

```
1 str_list = ['Hello Python!', 'Hello, World!']
2 for i, str_list in enumerate(str_list):
3     print(i, str_list)
```

0 Hello Python!

1 Hello, World!

In [155]:

```
1 str_list = ['Hello Python!', 'Hello, World!']
2 enumerate_list = enumerate(str_list)
3 print(list(enumerate_list))
```

[(0, 'Hello Python!'), (1, 'Hello, World!')]

filter()

It excludes items in an iterable object.

In [159]:

```
1 def filtering(data):
2     if data > 30:
3         return data
4
5 data = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
6 result = filter(filtering, data)
7 print(list(result))
```

[37, 1729]

globals()

It returns the current global symbol table as a dictionary.

In [39]:

```
1 globals()
```

Out[39]:

```
{'__name__': '__main__',
 '__doc__': 'Automatically created module for IPython interactive environment',
 '__package__': None,
 '__loader__': None,
 '__spec__': None,
 '__builtin__': <module 'builtins' (built-in)>,
 '__builtins__': <module 'builtins' (built-in)>,
 '_ih': [],
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(any(nlis))',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))',
 "nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))\nnlis.append(0, 'False')\nprint(nlis)",
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))\nnlis.append(0, False)\nprint(nlis)',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint
```

In [41]:

```
1 num = 37
2 globals()['num'] = 3.14
3 print(f'The number is {num}.')
```

The number is 3.14.

frozen()

It returns a frozenset object

In [36]:

```
1 nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 frozen_nlis = frozenset(nlis)
3 print('Frozen set is', frozen_nlis)
```

Frozen set is frozenset({0.577, 1.618, 2.718, 3.14, 1729, 37, 6, 28})

any()

It returns *True* if any iterable is *True*.

In [10]:

```
1 nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 print(nlis)
3 print(any(nlis))
4 nlis.clear()
5 print(nlis)
6 print(any(nlis))    # An empty list returns False
7 nlis.append(0)
8 print(nlis)
9 print(any(nlis))    # 0 in a list returns False
10 nlis.append(False)
11 print(nlis)
12 print(any(nlis))    # False in a list returns False
13 nlis.append(True)
14 print(nlis)
15 print(any(nlis))    # True in a list returns True
16 nlis.append(1)
17 print(nlis)
18 print(any(nlis))    # 1 in a list returns True
19 nlis.clear()
20 nlis.append(None)
21 print(nlis)
22 print(any(nlis))    # None in a list returns False
```

```
[0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
```

```
True
```

```
[]
```

```
False
```

```
[0]
```

```
False
```

```
[0, False]
```

```
False
```

```
[0, False, True]
```

```
True
```

```
[0, False, True, 1]
```

```
True
```

```
[None]
```

```
False
```

ascii()

It returns a string including a printable representation of an object and escapes non-ASCII characters in the string employing `\u`, `\x` or `\U` escapes

In [17]:

```
1 txt = 'Hello, Python!'
2 print(ascii(txt))
3 text = 'Hello, Pythän!'
4 print(ascii(text))
5 print('Hello, Pyth\xe4n!')
6 msg = 'Hellü, World!'
7 print(ascii(msg))
8 print('Hell\xfc, World!')
```

```
'Hello, Python!'
'Hello, Pyth\xe4n!'
Hello, Pythän!
'Hell\xfc, World!'
Hellü, World!
```

bytearray()

It returns a new array of bytes.

In [23]:

```
1 txt = 'Hello, Python!'
2 print(bytearray(txt, 'utf-8')) # String with encoding 'UTF-8'
3 int_num = 37
4 print(bytearray(int_num))
5 nlis = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34] # Fibonacci numbers
6 print(bytearray(nlis))
7 float_num = 3.14
8 print(bytearray(float_num)) # It returns TypeError
```

```
bytearray(b'Hello, Python!')
bytearray(b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
bytearray(b'\x00\x01\x01\x02\x03\x05\x08\r\x15')
```

```
-----
TypeError Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_16192\391563769.py in <module>
      6 print(bytearray(nlis))
      7 float_num = 3.14
----> 8 print(bytearray(float_num))
```

TypeError: cannot convert 'float' object to bytearray

hasattr()

It returns *True* if the specified object has the specified attribute (property/method).

In [47]:

```
1 class SpecialNumbers:
2     euler_constant = 0.577
3     euler_number = 2.718
4     pi = 3.14
5     golden_ratio = 1.618
6     msg = 'These numbers are special.'
7
8 special_numbers = SpecialNumbers()
9 print('The euler number is', hasattr(special_numbers, 'euler_number'))
10 print('The golden ratio is', hasattr(special_numbers, 'golden_ratio'))
11 print('The golden ratio is', hasattr(special_numbers, 'prime_number')) # Since there is no prime number, the output
```

The euler number is True

The golden ratio is True

The golden ratio is False

hash()

It returns the hash value of a specified object.

In [166]:

```
1 print(hash(3.14))
2 print(hash(0.577))
3 print(hash('Hello, Python!'))
4 print(hash(1729))
5 n_tuple = (0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729)
6 print(hash(n_tuple))
```

322818021289917443

1330471416316301312

-7855314544920281827

1729

-6529577050584256413

help()

Executes the built-in help system

In [167]:

```
1 help()
```

Welcome to Python 3.10's help utility!

If this is your first time using Python, you should definitely check out the tutorial on the internet at <https://docs.python.org/3.10/tutorial/>. (<https://docs.python.org/3.10/tutorial/>.)

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".

You are now leaving help and returning to the Python interpreter. If you want to ask for help on a particular object directly from the interpreter, you can type "help(object)". Executing "help('string')" has the same effect as typing a particular string at the help> prompt.

In [169]:

```
1 import pandas as pd
2 help(pd) # You can find more information about pandas.
```

Help on package pandas:

NAME
pandas

DESCRIPTION
pandas - a powerful data analysis and manipulation library for Python
=====

****pandas**** is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, ****real world**** data analysis in Python. Additionally, it has the broader goal of becoming ****the most powerful and flexible open source data analysis / manipulation tool available in any language****. It is already well on its way toward this goal.

Main Features

..

id()

Returns the id of an object

In [188]:

```
1 print(id('Hello, Python!'))
2 print(id(3.14))
3 print(id(1729))
4 special_nums_list = [0.577, 1.618, 2.718, 3.14, 28, 37, 1729]
5 print(id(special_nums_list))
6 special_nums_tuple = (0.577, 1.618, 2.718, 3.14, 28, 37, 1729)
7 print(id(special_nums_tuple))
8 special_nums_set = {0.577, 1.618, 2.718, 3.14, 28, 37, 1729}
9 print(id(special_nums_set))
10 special_nums_dict = {'Euler constant': 0.577, 'Golden ratio': 1.618,
11                      'Euler number': 2.718, 'PI number': 3.14,
12                      'Perfect number': 28, 'Prime number': 37,
13                      'Ramanujan Hardy number': 1729}
14 print(id(special_nums_dict))
```

```
1699639717104
1699636902256
1699636902896
1699639562944
1699639414208
1699639822816
1699639515264
```

eval()

This function evaluates and executes an expression.

In [26]:

```
1 num = int(input('Enter a number: '))
2 print(f'The entered number is {num}.')
3 print(eval('num*num'))
```

```
The entered number is 37.
1369
```

map()

It returns the specified iterator with the specified function applied to each item.

In [77]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 28, 37, 1729]
2 def division(number):
3     return number/number
4
5 division_number_iterator = map(division, special_nums)
6 divided_nums = list(division_number_iterator)
7 print(divided_nums)
8
9 # Similar codings can be made for other operations
```

```
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
```

len()

It returns the length of an object

In [54]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 print('The length of the list is', len(special_nums))
```

The length of the list is 8

In [59]:

```
1 # Calculate the average of values in the following list
2 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
3 count = 0
4 for i in special_nums:
5     count = count + i
6 print('The average of the values in the list is', count/len(special_nums))
```

The average of the values in the list is 226.00662499999999

min()

Returns the smallest item in an iterable

In [170]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 print(min(special_nums))
```

0.577

max()

Returns the largest item in an iterable

In [171]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 print(max(special_nums))
```

1729

sum()

To get the sum of numbers in a list

In [172]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 print(sum(special_nums))
```

1808.0529999999999

float()

It returns a floating point number.

In [28]:

```
1 int_num = 37
2 print(float(int_num))
3 float_num = 3.14
4 print(float(float_num))
5 txt = '2.718'
6 print(float(txt))
7 msg = 'Hello, Python!' # It returns a ValueError
8 print(float(msg))
```

37.0

3.14

2.718

```
-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_16192\162113112.py in <module>
      6 print(float(txt))
      7 msg = 'Hello, Python!' # It returns a ValueError
----> 8 print(float(msg))
```

ValueError: could not convert string to float: 'Hello, Python!'

locals()

It returns an updated dictionary of the current local symbol table.

In [68]:

```
1 locals()
```

Out[68]:

```
{'__name__': '__main__',
 '__doc__': 'Automatically created module for IPython interactive environment',
 '__package__': None,
 '__loader__': None,
 '__spec__': None,
 '__builtin__': <module 'builtins' (built-in)>,
 '__builtins__': <module 'builtins' (built-in)>,
 '_ih': [' ',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(any(nlis))',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))\nnlis.append(0, 'False')\nprint(nlis)",
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint(nlis)\nprint(any(nlis))\nnlis.append(0, False)\nprint(nlis)',
 'nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]\nprint(nlis)\nprint(any(nlis))\nnlis.clear()\nprint
```

In [70]:

```
1 def function():
2     variable = True
3     print(variable)
4     locals()['variable'] = False    # locals() dictionary may not change the information inside the locals table.
5     print(variable)
6
7 function()
```

True

True

In [75]:

```
1 def dict_1():
2     return locals()
3
4 def dict_2():
5     program = 'Python'
6     return locals()
7
8 print('If there is no locals(), it returns an empty dictionary', dict_1())
9 print('If there is locals(), it returns a dictionary', dict_2())
```

If there is no locals(), it returns an empty dictionary {}

If there is locals(), it returns a dictionary {'program': 'Python'}

format()

This function formats a specified value. **d**, **f**, and **b** are a type.

In [33]:

```
1 # integer format
2 int_num = 37
3 print(format(num, 'd'))
4 # float numbers
5 float_num = 2.7182818284
6 print(format(float_num, 'f'))
7 # binary format
8 num = 1729
9 print(format(num, 'b'))
```

```
37
2.718282
11011000001
```

hex()

Converts a number into a hexadecimal value

In [184]:

```
1 print(hex(6))
2 print(hex(37))
3 print(hex(1729))
```

```
0x6
0x25
0x6c1
```

input()

Allowing user input

In [219]:

```
1 txt = input('Enter a message: ')
2 print('The entered message is', txt)
```

The entered message is Hello, Python!

int()

Returns an integer number

In [223]:

```
1 num1 = int(6)
2 num2 = int(3.14)
3 num3 = int('28')
4 print(f'The numbers are {num1}, {num2},and {num3}.')
```

The numbers are 6, 3,and 28.

isinstance()

It checks if the object (first argument) is an instance or subclass of classinfo class (second argument).

In [226]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 result = isinstance(special_nums, list)
3 print(result)
```

True

In [225]:

```
1 class SpecialNumbers:
2     euler_constant = 0.577
3     euler_number = 2.718
4     pi = 3.14
5     golden_ratio = 1.618
6     msg = 'These numbers are very special'
7
8     def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
9         self.euler_constant = euler_constant
10        self.euler_number = euler_number
11        self.pi = pi
12        self.golden_ratio = golden_ratio
13        self.msg = msg
14
15 special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are very special.')
16 nums = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
17 print(isinstance(special_numbers, SpecialNumbers))
18 print(isinstance(nums, SpecialNumbers))
```

True

False

issubclass()

Checks if the class argument (first argument) is a subclass of classinfo class (second argument).

In [263]:

```
1 class Circle:
2     def __init__(circleType):
3         print('Circle is a ', circleType)
4
5 class Square(Circle):
6     def __init__(self):
7
8         Circle.__init__('square')
9
10 print(issubclass(Square, Circle))
11 print(issubclass(Square, list))
12 print(issubclass(Square, (list, Circle)))
13 print(issubclass(Circle, (list, Circle)))
```

True
False
True
True

iter()

It returns an iterator object.

In [52]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 special_nums_iter = iter(special_nums)
3 print('Euler constant is', next(special_nums_iter))
4 print('The golden ratio is', next(special_nums_iter))
5 print('Euler number is', next(special_nums_iter))
6 print('Pi number is', next(special_nums_iter))
7 print(next(special_nums_iter), 'is a perfect number.')
8 print(next(special_nums_iter), 'is a perfect number.')
9 print(next(special_nums_iter), 'is a special and prime number.')
10 print(next(special_nums_iter), 'is Ramanujan-Hardy number.')
```

Euler constant is 0.577
The golden ratio is 1.618
Euler number is 2.718
Pi number is 3.14
6 is a perfect number.
28 is a perfect number.
37 is a special and prime number.
1729 is Ramanujan-Hardy number.

object()

It returns a new object.

In [97]:

```
1 name= object()
2 print(type(name))
3 print(dir(name))
```

```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
```

oct()

It returns an octal string from the given integer number. The oct() function takes an integer number and returns its octal representation.

In [232]:

```
1 num = int(input('Enter a number:'))
2 print(f'The octal value of {num} us {oct(num)}.')
```

The octal value of 37 us 0o45.

In [235]:

```
1 # decimal to octal
2 print('oct(1729) is:', oct(1729))
3
4 # binary to octal
5 print('oct(0b101) is:', oct(0b101))
6
7 # hexadecimal to octal
8 print('oct(0XA) is:', oct(0XA))
```

oct(1729) is: 0o3301

oct(0b101) is: 0o5

oct(0XA) is: 0o12

list()

It creates a list in Python.

In [67]:

```

1 print(list())
2 txt = 'Python'
3 print(list(txt))
4 special_nums_set = {0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729}
5 print(list(special_nums_set))
6 special_nums_tuple = (0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729)
7 print(list(special_nums_tuple))
8 special_nums_dict = {'Euler constant': 0.577,
9                      'Golden ratio': 1.618,
10                     'Euler number': 2.718,
11                     'Pi number': 3.14,
12                     'Perfect number': 6,
13                     'Prime number': 37,
14                     'Ramanujan Hardy number': 1729}
15 print(list(special_nums_dict))

```

```

[]
['P', 'y', 't', 'h', 'o', 'n']
[0.577, 1.618, 2.718, 3.14, 1729, 37, 6, 28]
[0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
['Euler constant', 'Golden ratio', 'Euler number', 'Pi number', 'Perfect number', 'Prime number', 'Ramanujan Hardy number']

```

memoryview()

It returns a memory view object.

In [91]:

```

1 ba = bytearray('XYZ', 'utf-8')
2 mv = memoryview(ba)
3 print(mv)
4 print(mv[0])
5 print(mv[1])
6 print(mv[2])
7 print(bytes(mv[0:2]))
8 print(list(mv[:]))
9 print(set(mv[:]))
10 print(tuple(mv[:]))
11 mv[1] = 65 # 'Y' was replaced with 'A'
12 print(list(mv[:]))
13 print(ba)

```

```

<memory at 0x0000018BB24C5D80>
88
89
90
b'XY'
[88, 89, 90]
{88, 89, 90}
(88, 89, 90)
[88, 65, 90]
bytearray(b'XAZ')

```

In []:

1

In [218]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 number = iter(special_nums)    # Create an iteration
3 item = next(number)           # First item
4 print(item)
5 item = next(number)           # Second item
6 print(item)
7 item = next(number)           # Third item, etc
8 print(item)
9 item = next(number)
10 print(item)
11 item = next(number)
12 print(item)
13 item = next(number)
14 print(item)
15 item = next(number)
16 print(item)
17 item = next(number)
18 print(item)
```

```
0.577
1.618
2.718
3.14
6
28
37
1729
```

open()

It opens a file and returns a file object.

Character	Function
r	Open file for reading only. Starts reading from beginning of file. This default mode.
rb	Open a file for reading only in binary format. Starts reading from beginning of file.
r+	Open file for reading and writing. File pointer placed at beginning of the file.
w	Open file for writing only. File pointer placed at beginning of the file. Overwrites existing file and creates a new one if it does not exist.
wb	Same as w but opens in binary mode.
w+	Same as w but also allows to read from file.
wb+	Same as wb but also allows to read from file.
a	Open a file for appending. Starts writing at the end of file. Creates a new file if file does not exist.
ab	Same as a but in binary format. Creates a new file if file does not exist.
a+	Same a a but also open for reading.
ab+	Same a ab but also open for reading.

In [120]:

```
1 path = "authors.txt"
2 file = open(path, mode = 'r', encoding='utf-8')
3 print(file.name)
4 print(file.read())
```

```
authors.txt
English,Charles Severance
English,Sue Blumenberg
English,Elloitt Hauser
Spanish,Fernando Tardío Muñiz
```

In [122]:

```
1 # Open file using with
2 path = "authors.txt"
3 with open(path, "r") as file:
4     FileContent = file.read()
5     print(FileContent)
```

English,Charles Severance
English,Sue Blumenberg
English,Elloitt Hauser
Spanish,Fernando TardÃo MuÃ±iz

complex()

It returns a complex number.

In [138]:

```
1 print(complex(1))
2 print(complex(2, 2))
3 print(complex(3.14, 1.618))
```

(1+0j)

(2+2j)

(3.14+1.618j)

dir()

It returns a list of the specified object's properties and methods.

In [148]:

```

1 name = dir()
2 print(name)
3 print()
4 number = 3.14
5 print(dir(number))
6 print()
7 nlis = [3.14]
8 print(dir(nlis))
9 print()
10 nset = {3.14}
11 print(dir(nset))

```

```

['FileContent', 'In', 'Out', 'SpecialNumbers', '_', '_103', '_105', '_107', '_109', '_113', '_114', '_116', '_118',
'_119', '_144', '_39', '_68', '_92', '_98', '__', '__builtins__', '__builtins__', '__doc__', '__loader__', '_
_name__', '_package__', '_spec__', '_vsc_ipynb_file__', '_dh', '_i', '_i1', '_i10', '_i100', '_i101', '_i10
2', '_i103', '_i104', '_i105', '_i106', '_i107', '_i108', '_i109', '_i11', '_i110', '_i111', '_i112', '_i113', '_i114',
'_i115', '_i116', '_i117', '_i118', '_i119', '_i12', '_i120', '_i121', '_i122', '_i123', '_i124', '_i125', '_i126', '_i1
27', '_i128', '_i129', '_i13', '_i130', '_i131', '_i132', '_i133', '_i134', '_i135', '_i136', '_i137', '_i138', '_i139',
'_i14', '_i140', '_i141', '_i142', '_i143', '_i144', '_i145', '_i146', '_i147', '_i148', '_i15', '_i16', '_i17', '_i18',
'_i19', '_i2', '_i20', '_i21', '_i22', '_i23', '_i24', '_i25', '_i26', '_i27', '_i28', '_i29', '_i3', '_i30', '_i31', '_i32', '_
i33', '_i34', '_i35', '_i36', '_i37', '_i38', '_i39', '_i4', '_i40', '_i41', '_i42', '_i43', '_i44', '_i45', '_i46', '_i47', '_i
48', '_i49', '_i5', '_i50', '_i51', '_i52', '_i53', '_i54', '_i55', '_i56', '_i57', '_i58', '_i59', '_i6', '_i60', '_i61', '_i6
2', '_i63', '_i64', '_i65', '_i66', '_i67', '_i68', '_i69', '_i7', '_i70', '_i71', '_i72', '_i73', '_i74', '_i75', '_i76', '_i7
7', '_i78', '_i79', '_i8', '_i80', '_i81', '_i82', '_i83', '_i84', '_i85', '_i86', '_i87', '_i88', '_i89', '_i9', '_i90', '_i9
1', '_i92', '_i93', '_i94', '_i95', '_i96', '_i97', '_i98', '_i99', '_ih', '_ii', '_iii', '_oh', 'ba', 'count', 'dict_1', 'dict_
2', 'divided_nums', 'division', 'division_number_iterator', 'exit', 'file', 'float_num', 'frozen_nlis', 'function',
'get_ipython', 'i', 'int_num', 'msg', 'mv', 'name', 'nlis', 'num', 'number', 'os', 'path', 'python', 'quit', 'specia
l_numbers', 'special_nums', 'special_nums_dict', 'special_nums_iter', 'special_nums_set', 'special_nums
_tuple', 'sys', 'text', 'txt']

```

```

['_abs__', '_add__', '_bool__', '_ceil__', '_class__', '_delattr__', '_dir__', '_divmod__', '_doc_
__', '_eq__', '_float__', '_floor__', '_floordiv__', '_format__', '_ge__', '_getattr__', '_getfor
mat__', '_getnewargs__', '_gt__', '_hash__', '_init__', '_init_subclass__', '_int__', '_le__', '_lt_
__', '_mod__', '_mul__', '_ne__', '_neg__', '_new__', '_pos__', '_pow__', '_radd__', '_rdivmod_
__', '_reduce__', '_reduce_ex__', '_repr__', '_rfloordiv__', '_rmod__', '_rmul__', '_round__', '_r
pow__', '_rsub__', '_rtruediv__', '_set_format__', '_setattr__', '_sizeof__', '_str__', '_sub__', '_
subclasshook__', '_truediv__', '_trunc__', 'as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'imag', 'is_in
teger', 'real']

```

```

['_add__', '_class__', '_class_getitem__', '_contains__', '_delattr__', '_delitem__', '_dir__', '_do
c__', '_eq__', '_format__', '_ge__', '_getattr__', '_getitem__', '_gt__', '_hash__', '_iadd_
__', '_imul__', '_init__', '_init_subclass__', '_iter__', '_le__', '_len__', '_lt__', '_mul__', '_ne__',
'_new__', '_reduce__', '_reduce_ex__', '_repr__', '_reversed__', '_rmul__', '_setattr__', '_setit
em__', '_sizeof__', '_str__', '_subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'inse
rt', 'pop', 'remove', 'reverse', 'sort']

```

```

['_and__', '_class__', '_class_getitem__', '_contains__', '_delattr__', '_dir__', '_doc__', '_eq__',
'_format__', '_ge__', '_getattr__', '_gt__', '_hash__', '_iand__', '_init__', '_init_subclass_
__', '_ior__', '_isub__', '_iter__', '_ixor__', '_le__', '_len__', '_lt__', '_ne__', '_new__', '_or_
__', '_rand__', '_reduce__', '_reduce_ex__', '_repr__', '_ror__', '_rsub__', '_rxor__', '_setattr__', '_
sizeof__', '_str__', '_sub__', '_subclasshook__', '_xor__', 'add', 'clear', 'copy', 'difference', 'differen
ce_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remo
ve', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']

```

divmod()

It returns the quotient and the remainder when argument1 is divided by argument2.

In [152]:

```

1 print(divmod(3.14, 0.577))
2 print(divmod(9, 3))
3 print(divmod(12, 5))
4 print(divmod('Hello', 'Python!')) # It returns TypeError.

```

(5.0, 0.255000000000000034)

(3, 0)

(2, 2)

```

-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_16192\798993378.py in <module>
      2 print(divmod(9, 3))
      3 print(divmod(12, 5))
----> 4 print(divmod('Hello', 'Python!'))

```

TypeError: unsupported operand type(s) for divmod(): 'str' and 'str'

set()

It returns a new set object.

In [179]:

```

1 print(set())
2 print(set('3.15'))
3 print(set('Hello Python!'))
4 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
5 print(set(special_nums))
6 print(set(range(2, 9)))
7 special_nums_dict = {'Euler constant': 0.577, 'Golden ratio': 1.618, 'Euler number': 2.718, 'Pi number': 3.14, 'Perfect n
8 print(set(special_nums_dict))

```

set()

{'5', '1', '.', '3'}

{' ', 'o', 't', 'e', 'n', 'y', 'P', 'h', '!', 'H', 'l'}

{0.577, 1.618, 2.718, 3.14, 1729, 37, 6, 28}

{2, 3, 4, 5, 6, 7, 8}

{'Pi number', 'Euler number', 'Euler constant', 'Golden ratio', 'Perfect number'}

setattr()

Sets an attribute (property/method) of an object

In [195]:

```
1 class SpecialNumbers:
2     euler_constant = 0.0
3     euler_number = 0.0
4     pi = 0.0
5     golden_ratio = 0.0
6     msg = ""
7
8     def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
9         self.euler_constant = euler_constant
10        self.euler_number = euler_number
11        self.pi = pi
12        self.golden_ratio = golden_ratio
13        self.msg = msg
14
15 special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are special.')
16 print(special_numbers.euler_constant)
17 print(special_numbers.euler_number)
18 print(special_numbers.pi)
19 print(special_numbers.golden_ratio)
20 print(special_numbers.msg)
21 setattr(special_numbers, 'Ramanujan_Hardy_number', 1729)
22 print(special_numbers.Ramanujan_Hardy_number)
```

0.577

2.718

3.14

1.618

These numbers are special.

1729

slice()

Returns a slice object that is used to slice any sequence (string, tuple, list, range, or bytes).

In [210]:

```
1 print(slice(2.718))
2 print(slice(0.577, 1.618, 3.14))
3 msg = 'Hello, Python!'
4 sliced_msg = slice(5)
5 print(msg[sliced_msg])
6 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
7 sliced_list = slice(4)
8 print(special_nums[sliced_list])
9 sliced_list = slice(-1, -6, -2)
10 print(special_nums[sliced_list])
11 print(special_nums[0:4])      # Slicing with indexing
12 print(special_nums[-4:-1])
```

slice(None, 2.718, None)

slice(0.577, 1.618, 3.14)

Hello

[0.577, 1.618, 2.718, 3.14]

[1729, 28, 3.14]

[0.577, 1.618, 2.718, 3.14]

[6, 28, 37]

sorted()

Returns a sorted list

In [213]:

```
1 special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37]
2 print(sorted(special_nums))
3 txt = 'Hello, Python!'
4 print(sorted(txt))
```

[0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]

[' ', '!', ',', 'H', 'P', 'e', 'h', 'l', 'l', 'n', 'o', 'o', 't', 'y']

ord()

Convert an integer representing the Unicode of the specified character

Type *Markdown* and LaTeX: α^2

In [241]:

```
1 print(ord('9'))
2 print(ord('X'))
3 print(ord('W'))
4 print(ord('^'))
```

57

88

87

94

pow()

The pow() function returns the power of a number.

In [247]:

```
1 print(pow(2.718, 3.14))
2 print(pow(-25, -2))
3 print(pow(16, 3))
4 print(pow(-6, 2))
5 print(pow(6, -2))
```

```
23.09634618919156
0.0016
4096
36
0.027777777777777776
```

print()

It prints the given object to the standard output device (screen) or to the text stream file.

In [248]:

```
1 msg = 'Hello, Python!'
2 print(msg)
```

```
Hello, Python!
```

range()

Returns a sequence of numbers between the given start integer to the stop integer.

In [254]:

```
1 print(list(range(0)))
2 print(list(range(9)))
3 print(list(range(2, 9)))
4 for i in range(2, 9):
5     print(i)
```

```
[]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
[2, 3, 4, 5, 6, 7, 8]
2
3
4
5
6
7
8
```

reversed()

Returns the reversed iterator of the given sequence.

In [260]:

```
1 txt = 'Python'
2 print(list(reversed(txt)))
3 special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37]
4 print(list(reversed(special_nums)))
5 nums = range(6, 28)
6 print(list(reversed(nums)))
7 special_nums_tuple = (2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37)
8 print(list(reversed(special_nums_tuple)))
```

['n', 'o', 'h', 't', 'y', 'P']

[37, 6, 3.14, 28, 1.618, 0.577, 1729, 2.718]

[27, 26, 25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6]

[37, 6, 3.14, 28, 1.618, 0.577, 1729, 2.718]

round()

Returns a floating-point number rounded to the specified number of decimals.

In [261]:

```
1 print(round(3.14))
2 print(round(2.718))
3 print(round(0.577))
4 print(round(1.618))
5 print(round(1729))
```

3

3

1

2

1729

str()

Returns the string version of the given object.

In [268]:

```
1 num = 3.14
2 val = str(num)
3 print(val)
4 print(type(val))
```

3.14

<class 'str'>

tuple()

The tuple() builtin can be used to create tuples in Python. In Python, a tuple is an immutable sequence type. One of the ways of creating tuple is by using the tuple() construct.

In [271]:

```
1 special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37]
2 special_nums_tuple = tuple(special_nums)
3 print(special_nums_tuple)
4
5 txt = 'Hello, Python!'
6 txt_tuple = tuple(txt)
7 print(txt_tuple)
8
9 dictionary = {'A': 0.577, 'B': 2.718, 'C': 3.14}
10 dictionary_tuple = tuple(dictionary)
11 print(dictionary_tuple)
```

```
(2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37)
('H', 'e', 'l', 'l', 'o', ',', ' ', 'P', 'y', 't', 'h', 'o', 'n', '!')
('A', 'B', 'C')
```

type()

It either returns the type of the object or returns a new type object based on the arguments passed.

In [276]:

```

1 special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37]
2 special_nums_tuple = tuple(special_nums)
3 print(special_nums_tuple)
4 print(type(special_nums))
5 print()
6 txt = 'Hello, Python!'
7 txt_tuple = tuple(txt)
8 print(txt_tuple)
9 print(type(txt))
10 print()
11 dictionary = {'A': 0.577, 'B': 2.718, 'C': 3.14}
12 dictionary_tuple = tuple(dictionary)
13 print(dictionary_tuple)
14 print(type(dictionary))
15 print()
16 special_nums_set = {2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37}
17 special_nums_tuple = tuple(special_nums_set)
18 print(special_nums_tuple)
19 print(type(special_nums_set))
20 print()
21 class SpecialNumbers:
22     euler_constant = 0.577
23     euler_number = 2.718
24     pi = 3.14
25     golden_ratio = 1.618
26     msg = 'These numbers are very special'
27
28     def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
29         self.euler_constant = euler_constant
30         self.euler_number = euler_number
31         self.pi = pi
32         self.golden_ratio = golden_ratio
33         self.msg = msg
34
35 special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are very special.')
36 print(type(special_numbers))

```

(2.718, 1729, 0.577, 1.618, 28, 3.14, 6, 37)

<class 'list'>

('H', 'e', 'l', 'l', 'o', ',', ' ', 'P', 'y', 't', 'h', 'o', 'n', '!')

<class 'str'>

('A', 'B', 'C')

<class 'dict'>

(0.577, 1729, 2.718, 3.14, 1.618, 37, 6, 28)

<class 'set'>

<class '__main__.SpecialNumbers'>

vars()

The vars() function returns the **dict** attribute of the given object.

In [277]:

```
1 class SpecialNumbers:
2     euler_constant = 0.577
3     euler_number = 2.718
4     pi = 3.14
5     golden_ratio = 1.618
6     msg = 'These numbers are very special'
7
8     def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
9         self.euler_constant = euler_constant
10        self.euler_number = euler_number
11        self.pi = pi
12        self.golden_ratio = golden_ratio
13        self.msg = msg
14
15 special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are very special.')
16 print(vars(special_numbers))
```

```
{'euler_constant': 0.577, 'euler_number': 2.718, 'pi': 3.14, 'golden_ratio': 1.618, 'msg': 'These numbers are very special.'}
```

zip()

It takes iterables (can be zero or more), aggregates them in a tuple, and returns it.

In [283]:

```
1 special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 37]
2 special_nums_name = ['Euler number', 'Ramanujan-Hardy number', 'Euler constant', 'Golden ratio', 'Perfect number',
3 output = zip()
4 output_list = list(output)
5 print(output_list)
6 reel_output = zip(special_nums_name, special_nums)
7 reel_output_set = set(reel_output)
8 print(reel_output_set)
```

```
[]
{('Pi number', 3.14), ('Perfect number', 28), ('Euler number', 2.718), ('Euler constant', 0.577), ('Ramanujan-Hardy number', 1729), ('Golden ratio', 1.618), ('Prime number', 37)}
```

super()

Returns a proxy object (temporary object of the superclass) that allows us to access methods of the base class.

In [292]:

```
1 class SpecialNumbers(object):
2     def __init__(self, special_numbers):
3         print('6 and 28 are', special_numbers)
4
5 class PerfectNumbers(SpecialNumbers):
6     def __init__(self):
7
8         # call superclass
9         super().__init__('perfect numbers.')
10        print('These numbers are very special in matematik.')
11
12 nums = PerfectNumbers()
```

6 and 28 are perfect numbers.
These numbers are very special in matematik.

In [294]:

```
1 class Animal(object):
2     def __init__(self, AnimalName):
3         print(AnimalName, 'lives in a farm.')
4
5 class Cow(Animal):
6     def __init__(self):
7         print('Cow gives us milk.')
8         super().__init__('Cow')
9
10 result = Cow()
```

Cow gives us milk.
Cow lives in a farm.

import()

It is a function that is called by the import statement.

In [303]:

```
1 math = __import__('math', globals(), locals(), [], 0)
2 print(math.fabs(3.14))
3 print(math.fabs(-2.718))
4 print(math.pow(4, 3))
5 print(math.exp(-5))
6 print(math.log(2.718))
7 print(math.factorial(6))
```

3.14
2.718
64.0
0.006737946999085467
0.999896315728952
720

In [304]:

```
1 import math
2 print(math.fabs(3.14))
3 print(math.fabs(-2.718))
4 print(math.pow(4, 3))
5 print(math.exp(-5))
6 print(math.log(2.718))
7 print(math.factorial(6))
```

```
3.14
2.718
64.0
0.006737946999085467
0.999896315728952
720
```

Python Tutorial

Created by Mustafa Germec

12. Classes and Objects in Python

- Python is an **object-oriented programming language**.
- Unlike procedure-oriented programming, where the main emphasis is on functions, object-oriented programming stresses on objects.
- An **object** is simply a collection of data (variables) and methods (functions) that act on those data.
- Similarly, a **class** is a blueprint for that object.
- Like function definitions begin with the **def** keyword in Python, class definitions begin with a **class** keyword.
- The first string inside the class is called **docstring** and has a brief description of the class.
- Although not mandatory, this is highly recommended.

```

class Student:

    school_name = 'ABC School' ← Class Variables

    def __init__(self, name, age): ← Constructor to initialize
        self.name = name           Instance variables
        self.age = age

    @classmethod
    def change_school(cls, name): ← cls refer to the Class
        print(Student.school_name) ← Access Class Variables
        Student.school_name = name ← Modify Class Variables

jessa = Student('Jessa', 14)
Student.change_school('XYZ School') ← Call Class Method

```

Create a class

In [40]:

```

1 class Data:
2     num = 3.14
3
4 print(Data)

```

<class '__main__.Data'>

Create an object

In [41]:

```
1 class Data:
2     num = 3.14
3
4 var = Data()
5 print(var.num)
```

3.14

Function init()

In [43]:

```
1 class Data:
2     def __init__(self, euler_number, pi_number, golden_ratio):
3         self.euler_number = euler_number
4         self.pi_number = pi_number
5         self.golden_ratio = golden_ratio
6
7 val = Data(2.718, 3.14, 1.618)
8
9 print(val.euler_number)
10 print(val.golden_ratio)
11 print(val.pi_number)
```

2.718
1.618
3.14

Methods

In [45]:

```
1 class Data:
2     def __init__(self, euler_number, pi_number, golden_ratio):
3         self.euler_number = euler_number
4         self.pi_number = pi_number
5         self.golden_ratio = golden_ratio
6     def msg_function(self):
7         print("The euler number is", self.euler_number)
8         print("The golden ratio is", self.golden_ratio)
9         print("The pi number is", self.pi_number)
10
11 val = Data(2.718, 3.14, 1.618)
12 val.msg_function()
```

The euler number is 2.718
The golden ratio is 1.618
The pi number is 3.14

Self parameter

- The **self parameter** is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named **self**, you can call it whatever you like, but it has to be the **first parameter** of any function in the **class**.
- Check the following example:

In [46]:

```
1 """
2 The following codes are the same as the above codes under the title 'Methods'.
3 You see that the output is the same, but this codes contain 'classFirstParameter' instead of 'self'.
4 """
5 class Data:
6     def __init__(classFirstParameter, euler_number, pi_number, golden_ratio):
7         classFirstParameter.euler_number = euler_number
8         classFirstParameter.pi_number = pi_number
9         classFirstParameter.golden_ratio = golden_ratio
10
11     def msg_function(classFirstParameter):
12         print("The euler number is", classFirstParameter.euler_number)
13         print("The golden ratio is", classFirstParameter.golden_ratio)
14         print("The pi number is", classFirstParameter.pi_number)
15
16 val = Data(2.718, 3.14, 1.618)
17 val.msg_function()
```

The euler number is 2.718

The golden ratio is 1.618

The pi number is 3.14

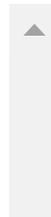
Creating a Class to draw a Rectangle

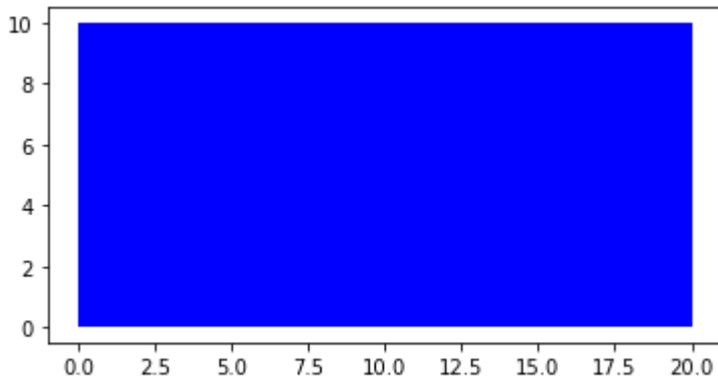
In [1]:

```
1 # Creating a class to draw a rectangle
2 class Rectangle(object):
3
4     # Contructor
5     def __init__(self, width, height, color):
6         self.width = width
7         self.height = height
8         self.color = color
9
10    # Method
11    def drawRectangle(self):
12        plt.gca().add_patch(plt.Rectangle((0, 0), self.width, self.height, fc=self.color))
13        plt.axis('scaled')
14        plt.show()
15
16    # import library to draw the Rectangle
17    import matplotlib.pyplot as plt
18    %matplotlib inline
19
20    # creating an object blue rectangle
21    one_Rectangle = Rectangle(20, 10, 'blue')
22
23    # Printing the object attribute width
24    print(one_Rectangle.width)
25
26    # Printing the object attribute height
27    print(one_Rectangle.height)
28
29    # Printing the object attribute color
30    print(one_Rectangle.color)
31
32    # Drawing the object
33    one_Rectangle.drawRectangle()
34
35    # Learning the methods that can be utilized on the object 'one_rectangle'
36    print(dir(one_Rectangle))
37
38    # We can change the properties of the rectangle
39    one_Rectangle.width = 15
40    one_Rectangle.height = 15
41    one_Rectangle.color = 'green'
42    one_Rectangle.drawRectangle()
43
44    # Using new variables, we can change the properties of the rectangle
45    two_Rectangle = Rectangle(100, 50, 'yellow')
46    two_Rectangle.drawRectangle()
47
48
```

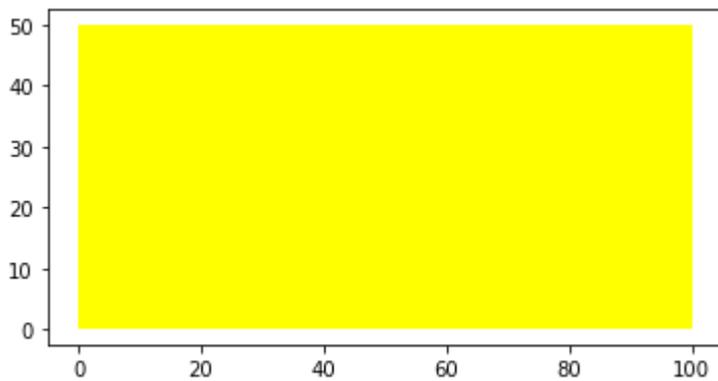
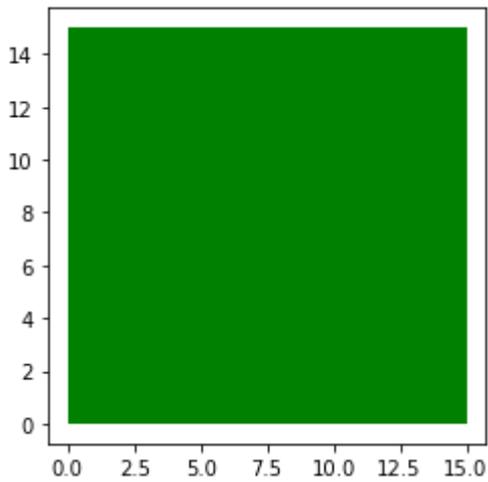
20

10





```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'color', 'drawRectangle', 'height', 'width']
```



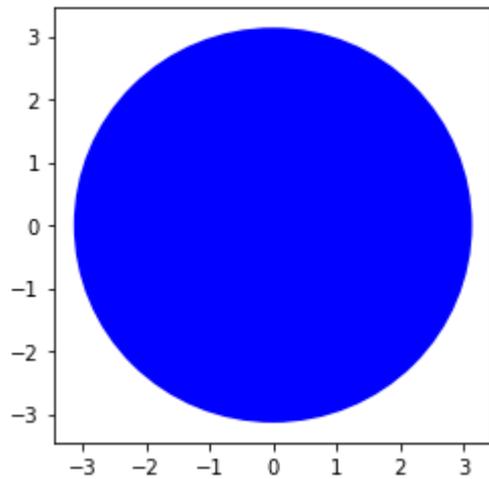
Creating a class to draw a circle

In [3]:

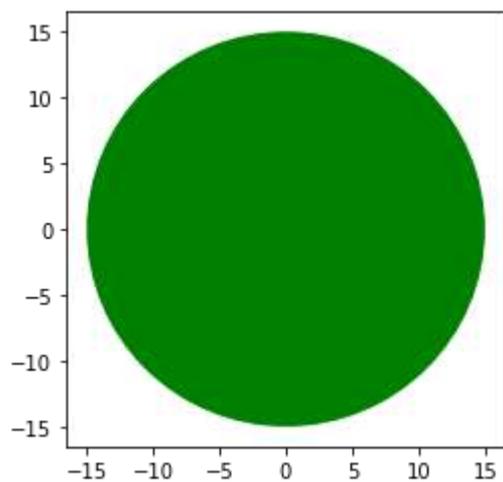
```
1 # Creating a class to draw a circle
2 class Circle(object):
3
4     # Contructor
5     def __init__(self, radius, color):
6         self.radius = radius
7         self.color = color
8
9     # Method
10    def increase_radius(self, r):
11        self.radius = self.radius + r
12        return self.radius
13
14    # Method
15    def drawCircle(self):
16        plt.gca().add_patch(plt.Circle((0, 0), self.radius, fc=self.color))
17        plt.axis('scaled')
18        plt.show()
19
20 # import library to draw the circle
21 import matplotlib.pyplot as plt
22 %matplotlib inline
23
24 # creating an object blue circle
25 one_Circle = Circle(3.14, 'blue')
26
27 # Printing the object attribute radius
28 print(one_Circle.radius)
29
30 # Printing the object attribute color
31 print(one_Circle.color)
32
33 # Drawing the object
34 one_Circle.drawCircle()
35
36 # Learning the methods that can be utilized on the object 'one_rectangle'
37 print(dir(one_Circle))
38
39 # We can change the properties of the rectangle
40 one_Circle.radius = 15
41 one_Circle.color = 'green'
42 one_Circle.drawCircle()
43
44 # Using new variables, we can change the properties of the rectangle
45 two_Circle = Circle(100, 'yellow')
46 print(two_Circle.radius)
47 print(two_Circle.color)
48 two_Circle.drawCircle()
49
50 # Changing the radius of the object
51 print('Before increment: ', one_Circle.radius)
52 one_Circle.drawCircle()
53
54 # Increment by 15 units
55 one_Circle.increase_radius(15)
56 print('Increase the radius by 15 units: ', one_Circle.radius)
57 one_Circle.drawCircle()
58
59 # Increment by 30 units
```

```
60 one_Circle.increase_radius(30)
61 print('Increase the radius by 30 units: ', one_Circle.radius)
62 one_Circle.drawCircle()
```

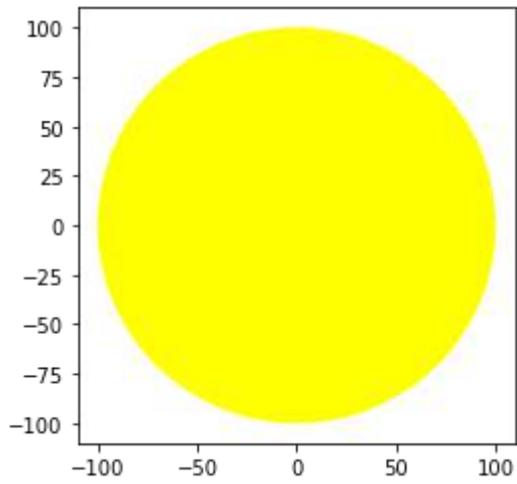
3.14
blue



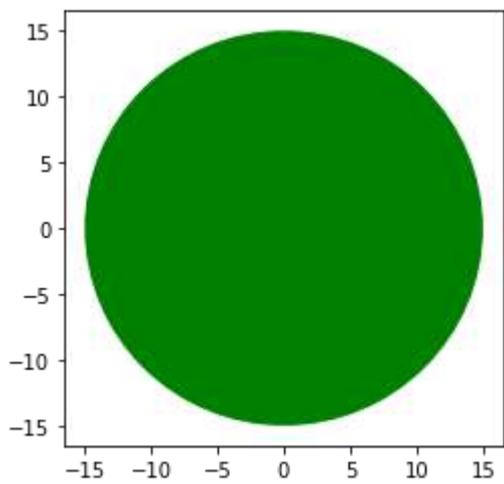
```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'color', 'drawCircle', 'increase_radius', 'radius']
```



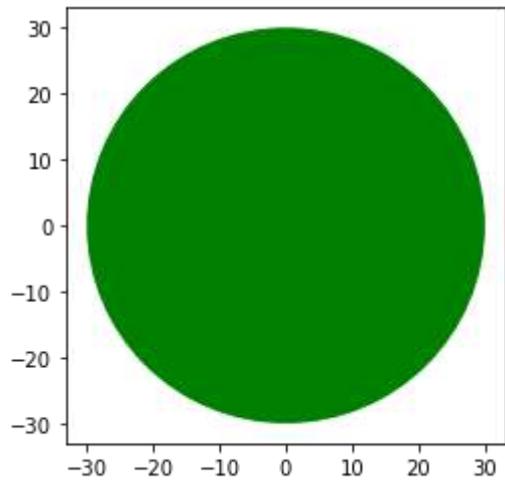
100
yellow



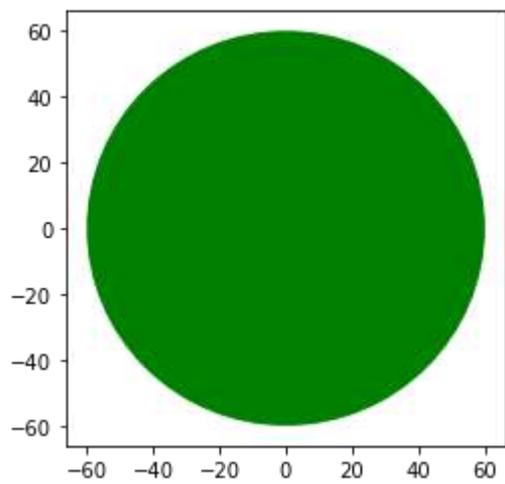
Before increment: 15



Increase the radius by 15 units: 30



Increase the radius by 30 units: 60



Some examples

In [36]:

```
1 class SpecialNumbers:
2     euler_constant = 0.577
3     euler_number = 2.718
4     pi_number = 3.14
5     golden_ratio = 1.618
6     msg = 'These numbers are special.'
7
8 special_numbers = SpecialNumbers()
9 print('The euler number is', getattr(special_numbers, 'euler_number'))
10 print('The golden ratio is', special_numbers.golden_ratio)
11 print('The pi number is', getattr(special_numbers, 'pi_number'))
12 print('The message is ', getattr(special_numbers, 'msg'))
```

The euler number is 2.718

The golden ratio is 1.618

The pi number is 3.14

The message is These numbers are special.

In [37]:

```

1 class SpecialNumbers:
2     euler_constant = 0.577
3     euler_number = 2.718
4     pi = 3.14
5     golden_ratio = 1.618
6     msg = 'These numbers are special.'
7
8     def parameter(self):
9         print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)
10
11 special_numbers = SpecialNumbers()
12 special_numbers.parameter()
13 delattr(SpecialNumbers, 'msg') # The code deleted the 'msg'.
14 special_numbers.parameter() # Since the code deleted the 'msg', it returns an AttributeError.

```

0.577 2.718 3.14 1.618 These numbers are special.

```

-----
AttributeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15364\3719874998.py in <module>
    12 special_numbers.parameter()
    13 delattr(SpecialNumbers, 'msg') # The code deleted the 'msg'.
--> 14 special_numbers.parameter() # Since the code deleted the 'msg', it returns an AttributeErr
or.

```

```

~\AppData\Local\Temp\ipykernel_15364\3719874998.py in parameter(self)
    7
    8 def parameter(self):
--> 9     print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)
    10
    11 special_numbers = SpecialNumbers()

```

AttributeError: 'SpecialNumbers' object has no attribute 'msg'

In [39]:

```

1 class ComplexNum:
2     def __init__(self, a, b):
3         self.a = a
4         self.b = b
5
6     def data(self):
7         print(f'{self.a}-{self.b}j')
8
9 var = ComplexNum(3.14, 1.618)
10 var.data()

```

3.14-1.618j

Create a Data Class

In [54]:

```
1 class Data:
2     def __init__(self, genus, species):
3         self.genus = genus
4         self.species = species
5
6     def microorganism(self):
7         print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')
8
9     #Use the Data class to create an object, and then execute the microorganism method
10    value = Data('Aspergillus', 'niger')
11    value.microorganism()
```

The name of a microorganism is in the form of Aspergillus niger.

Create a Child Class in Data Class

In [56]:

```
1 class Data:
2     def __init__(self, genus, species):
3         self.genus = genus
4         self.species = species
5
6     def microorganism(self):
7         print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')
8
9     class Recombinant(Data):
10        pass
11
12    value = Recombinant('Aspergillus', 'sojae')
13    value.microorganism()
```

The name of a microorganism is in the form of Aspergillus sojae.

Addition of init() Functions

In [4]:

```
1 class Data:
2     def __init__(self, genus, species):
3         self.genus = genus
4         self.species = species
5
6     def microorganism(self):
7         print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')
8
9 class Recombinant(Data):
10    def __init__(self, genus, species):
11        Data.__init__(self, genus, species)
12
13 value = Recombinant('Aspergillus', 'sojae')
14 value.microorganism()
```

The name of a microorganism is in the form of Aspergillus sojae.

Addition of super() Function

In [68]:

```
1 class SpecialNumbers(object):
2     def __init__(self, special_numbers):
3         print('6 and 28 are', special_numbers)
4
5 class PerfectNumbers(SpecialNumbers):
6     def __init__(self):
7
8         # call superclass
9         super().__init__('perfect numbers.')
10        print('These numbers are very special in matematik.')
11
12 nums = PerfectNumbers()
```

6 and 28 are perfect numbers.

These numbers are very special in matematik.

In [71]:

```
1 class Animal(object):
2     def __init__(self, AnimalName):
3         print(AnimalName, 'lives in a farm.')
4
5 class Cow(Animal):
6     def __init__(self):
7         print('Cow gives us milk.')
8         super().__init__('Cow')
9
10 result = Cow()
```

Cow gives us milk.

Cow lives in a farm.

In [60]:

```
1 class Data:
2     def __init__(self, genus, species):
3         self.genus = genus
4         self.species = species
5
6     def microorganism(self):
7         print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')
8
9 class Recombinant(Data):
10    def __init__(self, genus, species):
11        super().__init__(genus, species)    # 'self' statement in this line was deleted as different from the above codes
12
13 value = Recombinant('Aspergillus', 'sojae')
14 value.microorganism()
```

The name of a microorganism is in the form of Aspergillus sojae.

Addition of Properties under the super() Function

In [65]:

```
1 class Data:
2     def __init__(self, genus, species):
3         self.genus = genus
4         self.species = species
5
6     def microorganism(self):
7         print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')
8
9 class Recombinant(Data):
10    def __init__(self, genus, species):
11        super().__init__(genus, species)
12        self.activity = 2500    # This information was added as a Property
13
14 value = Recombinant('Aspergillus', 'sojae')
15 print(f'The enzyme activity increased to {value.activity} U/mL')
```

The enzyme activity increased to 2500 U/mL.

In [66]:

```

1 class Data:
2     def __init__(self, genus, species):
3         self.genus = genus
4         self.species = species
5
6     def microorganism(self):
7         print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')
8
9 class Recombinant(Data):
10    def __init__(self, genus, species, activity):
11        super().__init__(genus, species)
12        self.activity = activity      # This information was adedd as a Property
13
14    value = Recombinant('Aspergillus', 'sojae', 2500)
15    print(f'The enzyme activity increased to {value.activity} U/mL.')

```

The enzyme activity increased to 2500 U/mL.

Addition of Methods under the Child Class

In [67]:

```

1 class Data:
2     def __init__(self, genus, species):
3         self.genus = genus
4         self.species = species
5
6     def microorganism(self):
7         print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')
8
9 class Recombinant(Data):
10    def __init__(self, genus, species, activity):
11        super().__init__(genus, species)
12        self.activity = activity      # This information was adedd as a Property
13
14    def increment(self):
15        print(f'With this new recombinant {self.genus} {self.species} strain, the enzyme activity increased 2-times with {se
16
17    value = Recombinant('Aspergillus', 'sojae', 2500)
18    value.increment()

```

With this new recombinant *Aspergillus sojae* strain, the enzyme activity increased 2-times with 2500 U/mL.

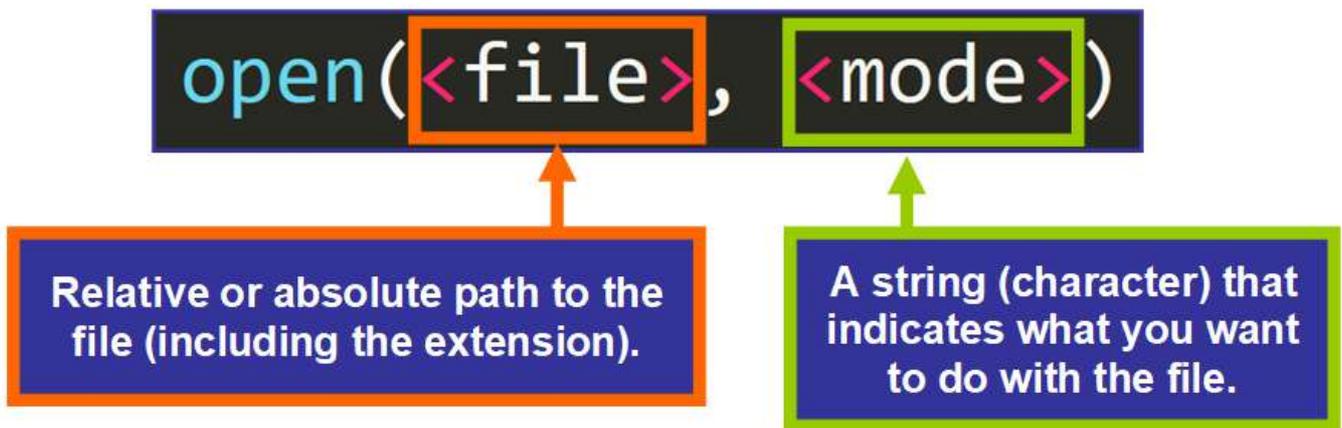
Python Tutorial

Created by Mustafa Germec, PhD

13. Reading Files in Python

To read a text file in Python, you follow these steps:

- First, open a text file for reading by using the **open()** function
- Second, read text from the text file using the file **read()**, **readline()**, or **readlines()** method of the file object.
- Third, close the file using the file **close()** method. This frees up resources and ensures consistency across different python versions.



Method	Description	Method	Description
<code>writable()</code>	Returns whether the file can be written to or not	<code>close()</code>	Closes the file
<code>readable()</code>	Returns whether the file stream can be read or not	<code>flush()</code>	Flushes the internal buffer
<code>read()</code>	Returns the file content	<code>seek()</code>	Change the file position
<code>readline()</code>	Returns one line from the file	<code>tell()</code>	Returns the current file
<code>readlines()</code>	Returns a list of lines from the file	<code>truncate()</code>	Resizes the file to a specified
<code>write()</code>	Writes the specified string to the file		
<code>writelines()</code>	Writes a list of strings to the file		

Reading file

In [2]:

```
1 # Reading the txt file
2 file_name = "pcr_file.txt"
3 file = open(file_name, "r")
4 content = file.read()
5 content
```

Out[2]:

I dedicate this book to Nancy Lier Cosgrove Mullis.
Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.
Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."
This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

In [3]:

```
1 # Printing the path of file
2 print(file.name)
3 # Printing the mode of file
4 print(file.mode)
5 # Printing the file with '\n' as a new file
6 print(content)
7 # Printing the type of file
8 print(type(content))
```

pcr_file.txt

r

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."

This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

<class 'str'>

In [4]:

```
1 # Close the file
2 file.close()
```

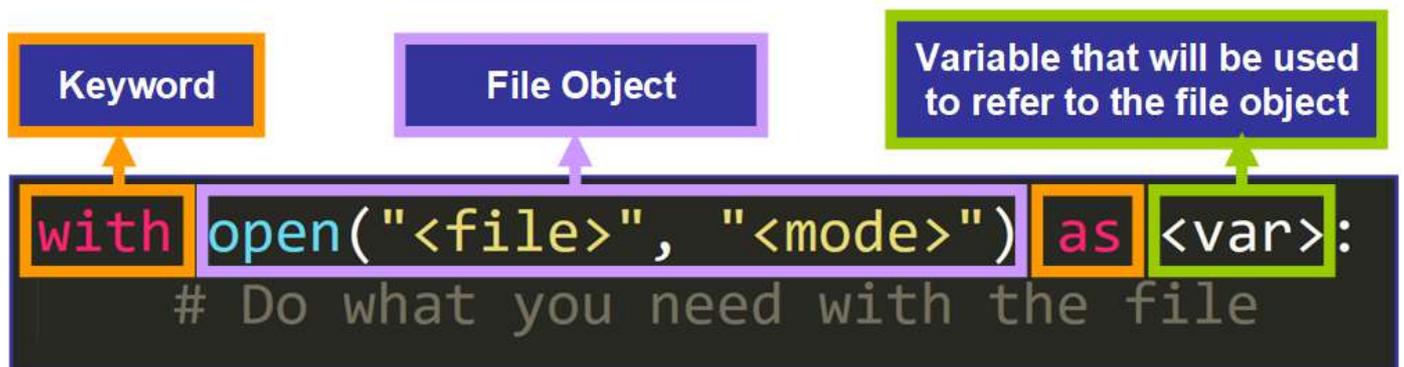
In [5]:

```
1 # Verification of the closed file
2 file.closed
```

Out[5]:

True

Another way to read a file



In [6]:

```
1 fname = 'pcr_file.txt'
2 with open(fname, 'r') as f:
3     content = f.read()
4     print(content)
```

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again. Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."

This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

In [7]:

```
1 # Verification of the closed file
2 f.closed
```

Out[7]:

True

In [8]:

```
1 # See the content of the file
2 print(content)
```

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again. Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."

This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

In [9]:

```
1 # Reading the first 20 characters in the text file
2 with open(fname, 'r') as f:
3     print(f.read(20))
```

I dedicate this book

In [10]:

```
1 # Reading certain amount of characters in the file
2 with open(fname, 'r') as f:
3     print(f.read(20))
4     print(f.read(20))
5     print(f.read(50))
6     print(f.read(100))
```

I dedicate this book

to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that

at we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have said

In [11]:

```
1 # Reading first line in the text file
2 with open(fname, 'r') as f:
3     print('The first line is: ', f.readline())
```

The first line is: I dedicate this book to Nancy Lier Cosgrove Mullis.

In [12]:

```
1 # Difference between read() and readline()
2 with open(fname, 'r') as f:
3     print(f.readline(20))
4     print(f.read(20)) # This code returns the next 20 characters in the line.
```

I dedicate this book
to Nancy Lier Cosgr

Loop usage in the text file

In [13]:

```
1 with open(fname, 'r') as f:
2     line_number = 1
3     for line in f:
4         print('Line number', str(line_number), ':', line)
5         line_number+=1
```

Line number 1 : I dedicate this book to Nancy Lier Cosgrove Mullis.

Line number 2 : Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Line number 3 : Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."

Line number 4 : This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

Line number 5 :

Methods

read(n) function

- Reads at most **n** bytes from the file if **n** is specified, else reads the entire file.
- Returns the retrieved bytes in the form of a string.

In [14]:

```
1 with open(fname, 'r') as f:  
2     print(f.read())
```

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."

This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

In [15]:

```
1 with open(fname, 'r') as f:  
2     print(f.read(30))
```

I dedicate this book to Nancy

readline() function

Reads one line at a time from the file in the form of string

In [16]:

```

1 with open(fname, 'r') as f:
2     file_list = f.readlines()
3     # Printing the first line
4     print(file_list[0])
5     # Printing the second line
6     print(file_list[1])
7     # Printing the third line
8     print(file_list[2])
9     # Printing the fourth line
10    print(file_list[3])

```

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."

This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

readlines() function

Reads all the lines from the file and returns a list of lines.

In [17]:

```

1 fname = r'C:/Users/test/Desktop/PROGRAMMING_WEB DEVELOPMENT/PYTHON_TUTORIAL/01. python_files_for_sh
2 with open(fname, 'r') as f:
3     content=f.readlines()
4     print(content)

```

```
[
'I dedicate this book to Nancy Lier Cosgrove Mullis.\n',
'Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.\n',
'Or maybe he would have just said, "If I'd had a woman like that, my books would not have been about despair."\n',
'This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.\n',
'\n']

```

strip() function

Removes the leading and trailing spaces from the given string.

In [18]:

```
1 with open(fname, 'r') as f:
2     len_file = 0
3     total_len_file = 0
4     for line in f:
5         # Total length of line in the text file
6         total_len_file = total_len_file+len(line)
7
8         # Length of the line after removing leading and trailing spaces
9         len_file = len_file+len(line.strip())
10    print(f'Total length of the line is {total_len_file}.')
11    print(f'The length of the line after removing leading and trailing spaces is {len_file}.')
12
```

Total length of the line is 1029.

The length of the line after removing leading and trailing spaces is 1024.

Size of the text file

In [19]:

```
1 with open(fname, 'r') as f:
2     str = ""
3     for line in f:
4         str+=line
5     print(f'The size of the text file is {len(str)}.')
```

The size of the text file is 1029.

Number of lines in the text

In [20]:

```
1 with open(fname, 'r') as f:
2     count = 0
3     for line in f:
4         count = count + 1
5     print(f'The number of lines in the text file is {count}.')
```

The number of lines in the text file is 5.

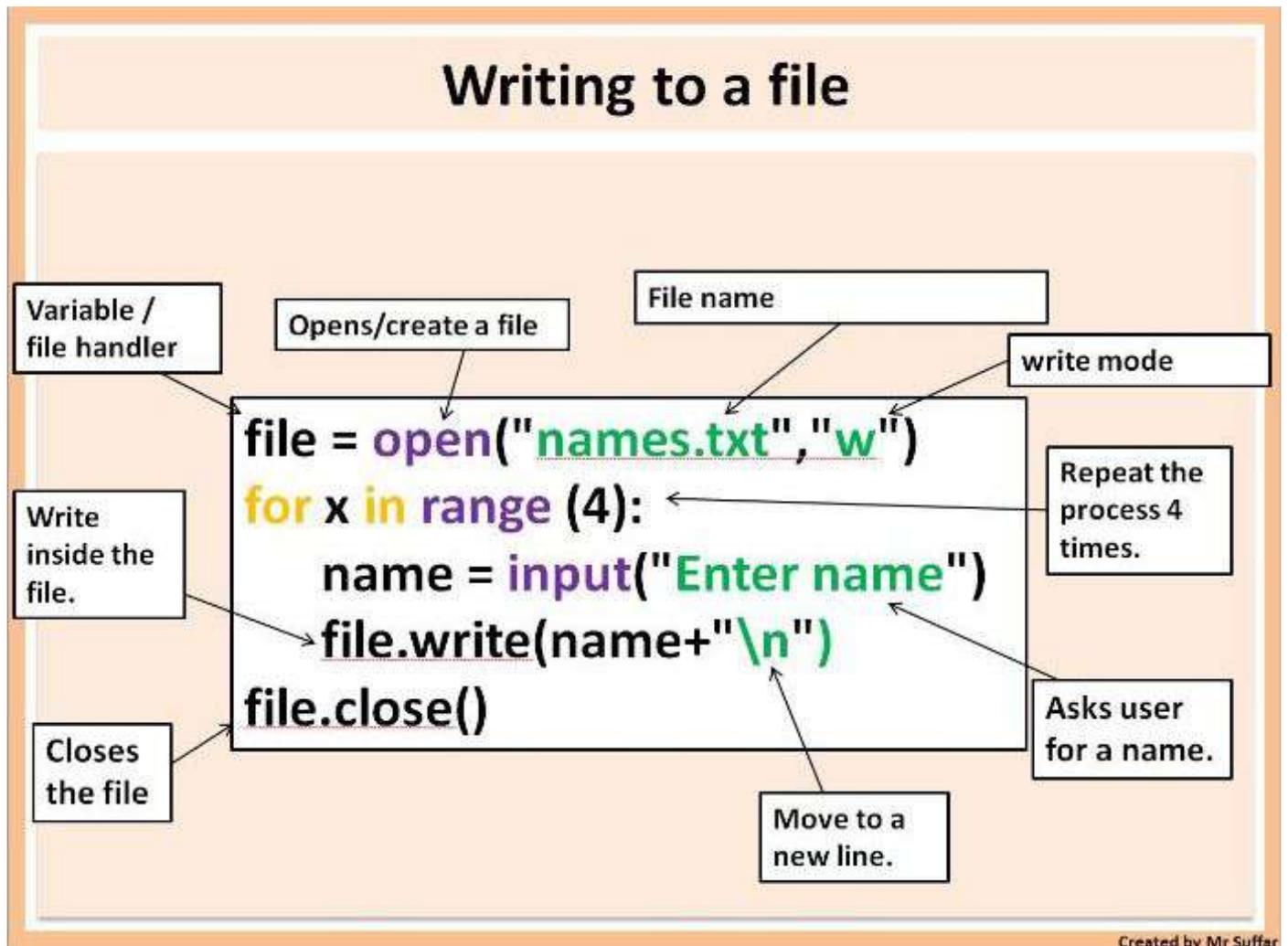
Python Tutorial

Created by Mustafa Germec, PhD

14. Writing Files in Python

To write to a text file in Python, you follow these steps:

- First, open the text file for writing (or appending) using the **open()** function.
- Second, write to the text file using the **write()** or **writelines()** method.
- Third, close the file using the **close()** method.



Character	Function
r	Open file for reading only. Starts reading from beginning of file. This default mode.
rb	Open a file for reading only in binary format. Starts reading from beginning of file.
r+	Open file for reading and writing. File pointer placed at beginning of the file.
w	Open file for writing only. File pointer placed at beginning of the file. Overwrites existing file and creates a new one if it does not exist.
wb	Same as w but opens in binary mode.
w+	Same as w but also allows to read from file.
wb+	Same as wb but also allows to read from file.
a	Open a file for appending. Starts writing at the end of file. Creates a new file if file does not exist.
ab	Same as a but in binary format. Creates a new file if file does not exist.
a+	Same as a but also open for reading.
ab+	Same as ab but also open for reading.

Writing files

In [17]:

```
1 # Writing lines to a file.
2 fname = 'pcr_file.txt'
3 with open(fname, 'w') as f:
4     f.write("I dedicate this book to Nancy Lier Cosgrove Mullis.\n")
5     f.write("Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. I
6     f.write("Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about
7     f.write("This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet
8     f.write("A feedback from Elle on the book\n")
9     f.write("This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free
```

In [18]:

```

1 # Checking the file whether it was written or not
2 with open(fname, 'r') as f:
3     content = f.read()
4     print(content)

```

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about despair.

'This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

A feedback from Elle on the book

This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free-wheeling adventure yarn that just happens to be the story of his life.

In [20]:

```

1 # Writing a list to a file
2 added_text = ["From The New Yorker\n",
3               "Entertaining ... [Mullis is] usefully cranky and combative, raising provocative questions about received truths from the scientific establishment.\n",
4               "From Chicago Sun-Times\n",
5               "One of the most unusual scientists of our times, a man who would be a joy to put under a microscope.\n",
6               "From Andrew Weil, M.D.\n",
7               "In this entertaining romp through diverse fields of inquiry, [Mullis] displays the openmindedness, eccentricity, brilliance, and general curmudgeonliness that make him the colorful character he is. His stories are engaging, informative, and fun.\n",
8               ]
9
10 fname = 'sample.txt'
11 with open(fname, 'w') as f:
12     for line in added_text:
13         print(line)
14         f.write(line)

```

From The New Yorker

Entertaining ... [Mullis is] usefully cranky and combative, raising provocative questions about received truths from the scientific establishment.

From Chicago Sun-Times

One of the most unusual scientists of our times, a man who would be a joy to put under a microscope.

From Andrew Weil, M.D.

In this entertaining romp through diverse fields of inquiry, [Mullis] displays the openmindedness, eccentricity, brilliance, and general curmudgeonliness that make him the colorful character he is. His stories are engaging, informative, and fun.

Appending files

In [24]:

```

1 # Writing and then reading the file
2 new_file = 'pcr_file.txt'
3 with open(new_file, 'w') as f:
4     f.write('Overright\n')
5 with open(new_file, 'r') as f:
6     print(f.read())

```

Overright

In [25]:

```

1 # Writing a new line to the text file
2 with open(new_file, 'a') as f:      # To append a new line to the text file, use 'a' in the syntax string
3     f.write("I dedicate this book to Nancy Lier Cosgrove Mullis.\n")
4     f.write("Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. I
5     f.write("Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about
6     f.write("This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet
7     f.write("A feedback from Elle on the book\n")
8     f.write("This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free
9
10 # Verification of the new lines in the text file
11 with open(new_file, 'r') as f:
12     print(f.read())

```

Overright

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about despair.

'This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

A feedback from Elle on the book

This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free-wheeling adventure yarn that just happens to be the story of his life.

Other modes

a+

Appending and Reading. Creates a new file, if none exists.

In [8]:

```
1 fname = 'pcr_file.txt'
2 with open(fname, 'a+') as f:
3     f.write("From F. Lee Bailey\n")
4     f.write("A very good book by a fascinating man.... [Mullis] enjoys an almost frighteningly brilliant mind and yet some
5     print(f.read())
```

In [9]:

```
1 # To verify the text file whether it is added or not
2 with open(fname, 'r') as f:
3     print(f.read())
```

Overright

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about despair.

'This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

A feedback from Elle on the book

This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free-wheeling adventure yarn that just happens to be the story of his life.

From F. Lee Bailey

A very good book by a fascinating man.... [Mullis] enjoys an almost frighteningly brilliant mind and yet somehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quick, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter moments aside, what he has to say is important.

tell() and seek() functions with a+

In [10]:

```

1  with open(fname, 'a+') as f:
2      print("First location: {}".format(f.tell()))  # it returns the current position in bytes
3
4      content = f.read()
5      if not content:
6          print('Read nothing.')
7      else:
8          print(f.read())
9
10     f.seek(0, 0)
11     """
12     seek() function is used to change the position of the File Handle to a given specific position.
13     File handle is like a cursor, which defines from where the data has to be read or written in the file.
14     Syntax: f.seek(offset, from_what), where f is file pointer
15     Parameters:
16     Offset: Number of positions to move forward
17     from_what: It defines point of reference.
18     Returns: Return the new absolute position.
19     The reference point is selected by the from_what argument. It accepts three values:
20         0: sets the reference point at the beginning of the file
21         1: sets the reference point at the current file position
22         2: sets the reference point at the end of the file
23     """
24     print('\nSecond location: {}'.format(f.tell()))
25     content = f.read()
26     if not content:
27         print('Read nothing.')
28     else:
29         print(content)
30     print('Location after reading: {}'.format(f.tell()))

```

First location: 1641

Read nothing.

Second location: 0

Overright

I dedicate this book to Nancy Lier Cosgrove Mullis.

Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about despair.

'This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

A feedback from Elle on the book

This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free-wheeling adventure yarn that just happens to be the story of his life.

From F. Lee Bailey

A very good book by a fascinating man.... [Mullis] enjoys an almost frighteningly brilliant mind and yet somehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quick, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter moments aside, what he has to say is important.

Location after reading: 1641

r+

Reading and writing. Cannot truncate the file.

In [13]:

```

1 with open(fname, 'r+') as f:
2     content=f.readlines()
3     f.seek(0,0)    # writing at the beginning of the file
4     f.write('From The San Diego Union-Tribune' + '\n')
5     f.write("Refreshing ... brashly confident ... indisputably entertaining." + "\n")
6     f.write("To my family..." + '\n')
7     f.seek(0,0)
8     print(f.read())

```

From The San Diego Union-Tribune

Refreshing ... brashly confident ... indisputably entertaining.

To my family...

...

s make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about despair.

'This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

A feedback from Elle on the book

This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free-wheeling adventure yarn that just happens to be the story of his life.

From F. Lee Bailey

A very good book by a fascinating man.... [Mullis] enjoys an almost frighteningly brilliant mind and yet somehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quick, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter moments aside, what he has to say is important.

Copy the file

In [14]:

```

1 # Let's copy the text file 'pcr_file.txt' to another one 'pcr_file_1.txt'
2 fname = 'pcr_file.txt'
3 with open(fname, 'r') as f_reading:
4     with open('pcr_file_1.txt', 'w') as f_writing:
5         for line in f_reading:
6             f_writing.write(line)

```

In [15]:

```

1 # For the verification, execute the following codes
2 fname = 'pcr_file_1.txt'
3 with open(fname, 'r') as f:
4     print(f.read())
5
6 # Now, there are 2 files from the same file content.
```

From The San Diego Union-Tribune

Refreshing ... brashly confident ... indisputably entertaining.

To my family...

...

s make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have noted that at least one man, someday, might get very lucky, and make his own heaven out of one of the people around him. She will be his morning and his evening star, shining with the brightest and the softest light in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless once again.

Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about despair.

'This book is not about despair. It is about a little bit of a lot of things, and, if not a single one of them is wet with sadness, it is not due to my lack of depth; it is due to a year of Nancy, and the prospect of never again being without her.

A feedback from Elle on the book

This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free-wheeling adventure yarn that just happens to be the story of his life.

From F. Lee Bailey

A very good book by a fascinating man.... [Mullis] enjoys an almost frighteningly brilliant mind and yet somehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quick, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter moments aside, what he has to say is important.

Some examples

In [36]:

```

1 # Writing the student names into a file
2 fname = open(r'student_name.txt', 'w')
3 for i in range(3):
4     name = input('Enter a student name: ')
5     fname.write(name)
6     fname.write('\n')      # To write names as a new line
7 fname = open(r'student_name.txt', 'r')
8 for line in fname:
9     print(line)
10 fname.close()
```

Daniela

Axel

Leonardo

In [40]:

```
1 # To write the file
2 fname = open(r'student_name.txt', 'w')
3 name_list = []
4 for i in range(3):
5     name = input('Enter a student name: ')
6     name_list.append(name + '\n')
7 fname.writelines(name_list)
8
9 # To read the file
10 fname = open(r'student_name.txt', 'r')
11 for line in fname:
12     print(line)
13 fname.close()
```

Daniel

Axel

Leonardo

In [41]:

```
1 lines = ['Hello, World!', 'Hi, Python!']
2 with open('new_file.txt', 'w') as f:
3     for line in lines:
4         f.write(line)
5         f.write('\n')
6
7 with open('new_file.txt', 'r') as f:
8     print(f.read())
```

Hello, World!

Hi, Python!

In [43]:

```
1 # Add more lines into the file
2 more_lines = ['Hi, Sun!', 'Hello, Summer!', 'Hi, See!']
3 with open('new_file.txt', 'a') as f:
4     f.writelines('\n'.join(more_lines))
5
6 with open('new_file.txt', 'r') as f:
7     print(f.read())
8
9 f.close()
```

Hello, World!

Hi, Python!

Hi, Sun!

Hello, Summer!

Hi, See!Hi, Sun!

Hello, Summer!

Hi, See!

Python Tutorial

Created by Mustafa Germec

15. Strings Operators and Functions in Python

Special String Operators in Python

indexing	<pre>>>>a="HELLO" >>>print(a[0]) >>>H >>>print(a[-1]) >>>0</pre>	<ul style="list-style-type: none"> ❖ Positive indexing helps in accessing the string from the beginning ❖ Negative subscript helps in accessing the string from the end.
Slicing:	<pre>Print[0:4] – HELL Print[:3] – HEL Print[0:]- HELLO</pre>	<p>The Slice[start : stop] operator extracts sub string from the strings. A segment of a string is called a slice.</p>
Concatenation	<pre>a="save" b="earth" >>>print(a+b) saveearth</pre>	<p>The + operator joins the text on both sides of the operator.</p>
Repetitions:	<pre>a="animalar " >>>print(3*a) animalaranimalar animalar</pre>	<p>The * operator repeats the string on the left hand side times the value on right hand side.</p>
Membership:	<pre>>>> s="good morning" >>>"m" in s True >>> "a" not in s True</pre>	<p>Using membership operators to check a particular character is in string or not. Returns true if present</p>

Strings are unchangeable

In [1]:

```
1 string_hello = 'Hello'
2 string_python = 'Python!'
3 print(string_hello)
4 print(string_python)
```

Hello
Python!

Deleting the items in a string is not supported since strings are immutable

It returns a `TypeError`. However, the whole string can be deleted. When it is, it returns a `NameError`.

In [6]:

```
1 text = 'Python is a programming language.'
2 print(text)
3 del text[1]
4 print(text)
```

Python is a programming language.

```
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18660\4179913587.py in <module>
      1 text = 'Python is a programming language.'
      2 print(text)
----> 3 del text[1]
      4 print(text)
```

TypeError: 'str' object doesn't support item deletion

In [7]:

```
1 text = 'Python is a programming language.'
2 print(text)
3 del text
4 print(text)
```

Python is a programming language.

```
-----
NameError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18660\1747540232.py in <module>
      2 print(text)
      3 del text
----> 4 print(text)
```

NameError: name 'text' is not defined

Concatenation of strings

It combines two or more strings using the sign '+' to form a new string.

In [9]:

```
1 text1 = 'Hello '  
2 text2 = 'Python!'  
3 new_text = text1 + text2  
4 print(new_text)
```

Hello Python!

Appending (+=) adds a new string the the end of the current string

In [10]:

```
1 text1 = 'Hello '  
2 text2 = 'Python!'  
3 text1+=text2  
4 print(text1)
```

Hello Python!

To repeat a string, the multiplication (*) operator is used

In [12]:

```
1 text = 'Hello, Python! '  
2 print(text*4)
```

Hello, Python! Hello, Python! Hello, Python! Hello, Python!

Accessing the item by indexing

In [21]:

```
1 text = 'Hello, Python!'  
2 print(text[0:5])    # Positive indexing  
3 print(text[4])  
4 print(text[-7:])   # Negative indexing  
5 print(text[-7:-1])
```

Hello

o

Python!

Python

Striding in slicing

The third parameter specifies the stride, which refers to how many characters to move forward after the first character is retrieved from the string.

In [29]:

```
1 text = 'Hello, Python!'
2 print(len(text))
3 print(text[:14])    # Default stride value is 1.
4 print(text[0:14:2])
5 print(text[::-3])
```

```
14
Hello, Python!
Hlo yhn
HI tn
```

Reverse string

The stride value is equal to -1 if a reverse string is wanted to obtain

In [30]:

```
1 text = 'Hello, Python!'
2 print(text[::-1])
```

```
!nohtyP ,olleH
```

in and not in

- **in** returns *True* when the character or word is in the given string, otherwise *False*.
- **not in** returns *False* when the character or word is in the given string, otherwise *True*.

In [130]:

```
1 text = 'Hello, Python!'
2 print('H' in text)
3 print('H' not in text)
4 print('c' not in text)
5 print('c' in text)
```

```
True
False
True
False
```

String Functions in Python

You can find some useful functions from the below table.

Method	Description
<code>capitalize()</code>	Capitalizes first letter of string
<code>find(str, beg=0 end=len(string))</code>	Determine if <code>str</code> occurs in <code>string</code> or in a substring of <code>string</code> if starting index <code>beg</code> and ending index <code>end</code> are given returns index if found and -1 otherwise
<code>isalnum()</code>	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise
<code>isalpha()</code>	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise
<code>isdigit()</code>	Returns true if string contains only digits and false otherwise
<code>islower()</code>	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise
<code>isnumeric()</code>	Returns true if a unicode string contains only numeric characters and false otherwise
<code>isspace()</code>	Returns true if string contains only whitespace characters and false otherwise
<code>istitle()</code>	Returns true if string is properly "titlecased" and false otherwise
<code>isupper()</code>	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise
<code>join(seq)</code>	Merges (concatenates) the string representations of elements in sequence <code>seq</code> into a string, with separator string
<code>len(string)</code>	Returns the length of the string
<code>lower()</code>	Converts all uppercase letters in string to lowercase
<code>max(str)</code>	Returns the max alphabetical character from the string <code>str</code>
<code>min(str)</code>	Returns the min alphabetical character from the string <code>str</code>
<code>replace(old, new [, max])</code>	Replaces all occurrences of <code>old</code> in string with <code>new</code> or at most <code>max</code> occurrences if <code>max</code> given
<code>split(str="", num=string.count(str))</code> <code>splitlines(num=string.count('\n'))</code>	Splits string according to delimiter <code>str</code> (space if not provided) and returns list of substrings; split into at most <code>num</code> substrings if given
<code>startswith(str, beg=0,end=len(string))</code>	Determines if string or a substring of string (if starting index <code>beg</code> and ending index <code>end</code> are given) starts with substring <code>str</code> ; returns true if so and false otherwise
<code>strip([chars])</code>	Performs both <code>lstrip()</code> and <code>rstrip()</code> on string
<code>swapcase()</code>	Inverts case for all letters in string
<code>title()</code>	Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase
<code>upper()</code>	Converts lowercase letters in string to uppercase
<code>isdecimal()</code>	Returns true if a unicode string contains only decimal characters and false otherwise

capitalize() function

It converts the first character of the string into uppercase.

In [31]:

```
1 text = 'hello, python!'
2 print(f'Before capitalizing: {text}')
3 text = text.capitalize()
4 print(f'After capitalizing: {text}')
```

Before capitalizing: hello, python!

After capitalizing: Hello, python!

casefold() function

It converts the characters in the certain string into lowercase.

In [32]:

```

1 text = 'Hello, Python!'
2 print(f'Before casefold: {text}')
3 text = text.casefold()
4 print(f'After casefold: {text}')

```

Before casefold: Hello, Python!

After casefold: hello, python!

center() function

It will center align the string, using a specified character (space is default) as the fill character.

In [43]:

```

1 text = 'Hello, Python!'
2 print(f'Before center() function: {text}')
3 text = text.center(50)
4 print(f'After center() function: {text}')
5 new_text = 'Hi, Python!'
6 new_text = new_text.center(50, '-')
7 print(f'After center() function: {new_text}')

```

Before center() function: Hello, Python!

After center() function: Hello, Python!

After center() function: -----Hi, Python!-----

count() function

It returns the number of a certain characters in a string.

In [45]:

```

1 text = 'Hello, Python!'
2 print(f"The number of the character 'o' in the string is {text.count('o')}." )

```

The number of the character 'o' in the string is 2.

endswith() function

It returns *True* if the strings ends with a certain value.

In [49]:

```

1 text = 'Hello, Python!'
2 text = text.endswith('Python!')
3 print(text)
4 new_text = 'Hi, Python!'
5 new_text = new_text.endswith('World!')
6 print(new_text)

```

True

False

find() function

It investigates the string for a certain value and returns the position of where it was found.

In [62]:

```
1 text = 'Hello, Python!'
2 print(text.find('Python'))
3 print(text.find('World', 0, 14))    # It returns -1 if the value is not found.
```

```
7
-1
```

format() function

- It formats the specified value(s) and insert them inside the string's placeholder.
- The placeholder is defined using curly brackets: {}.

In [66]:

```
1 text = 'Hello {} and Hi {}'.format('World!', 'Python!')
2 print(text)
3 text = 'Hello {world} and Hi {python}'.format(world='World!', python='Python!')
4 print(text)
5 text = 'Hello {0} and Hi {1}'.format('World!', 'Python!')
6 print(text)
7 text = 'Hello {1} and Hi {0}'.format('World!', 'Python!')
8 print(text)
```

```
Hello World! and Hi Python!
Hello World! and Hi Python!
Hello World! and Hi Python!
Hello Python! and Hi World!
```

index() function

It examines the string for a certain value and returns the position of where it was found.

In [71]:

```

1 text = 'Hello, Python!'
2 print(text.index('Python!'))
3 print(text.index('Hello'))
4 print(text.index('Hi')) # If the value is not found, it returns a 'ValueError'

```

7
0

```

-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18660\3544138795.py in <module>
      2 print(text.index('Python!'))
      3 print(text.index('Hello'))
----> 4 print(text.index('Hi')) # If the value is not found, it returns a 'ValueError'

```

ValueError: substring not found

isalnum() function

It returns *True* if all characters in the string are alphanumeric.

In [76]:

```

1 text = 'Hello, Python!'
2 print(text.isalnum())
3 msg = 'Hello1358'
4 print(msg.isalnum())

```

False
True

isalpha() function

- It returns *True* if all characters in the string are alphabets.
- White spaces are not considered as alphabets and thus it returns *False*.

In [80]:

```

1 text = 'Hello'
2 print(text.isalpha())
3 text = 'Hello1358' # The text contains numbers.
4 print(text.isalpha())
5 text = 'Hello Python!' # The text contains a white space.
6 print(text.isalpha())

```

True
False
False

isdecimal() function

It returns *True* if all the characters in the given string are decimal numbers (0-9).

In [82]:

```
1 text = 'Hello'
2 print(text.isdecimal())
3 numbered_text = '011235813'
4 print(numbered_text.isdecimal())
```

False

True

isdigit() function

This function returns *True* if all the characters in the string and the Unicode characters are digits.

In [83]:

```
1 numbered_text = '011235813'
2 print(numbered_text.isdigit())
```

True

isidentifier() function

It returns *True* if the string is a valid identifier, on the contrary *False*.

In [85]:

```
1 numbered_text = '011235813'
2 print(numbered_text.isidentifier())
3 variable = 'numbered_text'
4 print(variable.isidentifier())
```

False

True

isprintable() function

It returns *True* if all the characters in the string are printable features.

In [89]:

```
1 text = 'Hello, Python!'
2 print(text.isprintable())
3 new_text = 'Hello, \n Python!'
4 print(new_text.isprintable())
5 space = ' '
6 print(space.isprintable())
```

True

False

True

isspace() function

It returns *True* if all the characters in the string are whitespaces.

In [90]:

```
1 text = 'Hello, Python!'
2 print(text.isspace())
3 space = ' '
4 print(space.isspace())
```

False

True

islower() and lower() functions

- The function **islower()** returns *True* if all the characters in the string are lower case, on the contrary *False*.
- The function **lower()** converts the certain string to lower case.

In [94]:

```
1 text = 'Hello, Python!'
2 print(text.islower())
3 text = text.lower()      # It converts to lower case all the characters in the string.
4 print(text.islower())   # Now, it returns True.
```

False

True

isupper() and upper() functions

- The function **isupper()** returns *True* if all the characters in the string are upper case, on the contrary *False*.
- The function **upper()** converts the string to uppercase.

In [95]:

```
1 text = 'Hello, Python!'
2 print(text.isupper())
3 text = text.upper()      # It converts to upper case all the characters in the string.
4 print(text.isupper())   # Now, it returns True.
```

False

True

join() function

- It takes all items in an iterable and joins them into one string.
- A string must be specified as the separator.

In [100]:

```

1 text_list = ['Hello', 'World', 'Hi', 'Python']
2 print('#'. join(text_list))
3 text_tuple = ('Hello', 'World', 'Hi', 'Python')
4 print('+'. join(text_tuple))
5 text_set = {'Hello', 'World', 'Hi', 'Python'}
6 print('--'. join(text_set))
7 text_dict = {'val1': 'Hello', 'val2': 'World', 'val3': 'Hi', 'val4': 'Python'}
8 print('--'. join(text_dict))

```

```

Hello#World#Hi#Python
Hello+World+Hi+Python
Hello--Python--Hi--World
val1--val2--val3--val4

```

ljust() function

It returns the left justified version of the certain string.

In [119]:

```

1 text = 'Python'
2 text = text.ljust(30, '-')
3 print(text, 'is my favorite programming language.')

```

```
Python----- is my favorite programming language.
```

rjust() function

It returns the right justified version of the certain string.

In [120]:

```

1 text = 'Python'
2 text = text.rjust(30, '-')
3 print(text, 'is my favorite programming language.')

```

```
-----Python is my favorite programming language.
```

lstrip() function

It removes characters from the left based on the argument (a string specifying the set of characters to be removed).

In [103]:

```

1 text = '    Hello Python!    '
2 print(text.lstrip())    # It did not delete the white spaces in the right side.

```

```
Hello Python!
```

rstrip() function

It removes characters from the right based on the argument (a string specifying the set of characters to be removed).

In [104]:

```
1 text = '    Hello Python!    '  
2 print(text.rstrip())    # It did not delete the white spaces in the left side.
```

Hello Python!

strip() function

- It removes or truncates the given characters from the beginning and the end of the original string.
- The default behavior of the **strip()** method is to remove the whitespace from the beginning and at the end of the string.

In [105]:

```
1 text = '    Hello Python!    '  
2 print(text.strip())    # It deleted the white spaces in the both side.
```

Hello Python!

replace() function

Replaces a specified phrase with another specified phrase.

In [106]:

```
1 text = 'JavaScript is a programming language.'  
2 print(text)  
3 modified_text = text.replace('JavaScript', 'Python', 1)  
4 print(modified_text)
```

JavaScript is a programming language.

Python is a programming language.

In [107]:

```
1 text = 'Jython is a programming language.'  
2 print(text)  
3 modified_text = text.replace('J', 'P')  
4 print(modified_text)
```

Jython is a programming language.

Python is a programming language.

partition() function

- It searches for a specified string, and splits the string into a tuple containing three elements.
- The first element contains the part before the specified string.
- The second element contains the specified string.
- The third element contains the part after the string.

In [114]:

```
1 text = 'Hello World!, Hi Python!'
2 print(text.partition('Hi'))
```

('Hello World!, ', 'Hi', ' Python!')

rfind() function

- The **rfind()** method finds the last occurrence of the specified value.
- The **rfind()** method returns **-1** if the value is not found.
- The **rfind()** method is almost **the same as the rindex() method**.

In [125]:

```
1 text = 'Hello, Python is my favorite programming language.'
2 print(f"'Python' is in the position {text.rfind('Python')}")
3 print(f"'my' is in the position {text.rfind('my')}")
4 print(f"'close' is in the position {text.rfind('close')}")
```

'Python' is in the position 7.

'my' is in the position 17.

'close' is in the position -1.

rindex() function

- The **rindex()** method finds the last occurrence of the specified value.
- The **rindex()** method raises a **ValueError** exception if the value is not found.
- The **rindex()** method is almost **the same as the rfind() method**.

In [126]:

```
1 text = 'Hello, Python is my favorite programming language.'
2 print(f"'Python' is in the position {text.rindex('Python')}")
3 print(f"'my' is in the position {text.rindex('my')}")
4 print(f"'close' is in the position {text.rindex('close')}")
```

'Python' is in the position 7.

'my' is in the position 17.

```
-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_18660\2547564577.py in <module>
      2 print(f"'Python' is in the position {text.rindex('Python')}")
      3 print(f"'my' is in the position {text.rindex('my')}")
----> 4 print(f"'close' is in the position {text.rindex('close')}")
```

ValueError: substring not found

swapcase() function

This function converts the uppercase characters into lowercase and vice versa.

In [116]:

```
1 text = 'Hello Python!'
2 print(text.swapcase())
3 text = 'hELLO pYTHON!'
4 print(text.swapcase())
```

hELLO pYTHON!

Hello Python!

title() function

This function converts the first character in the given string into uppercase.

In [117]:

```
1 text = 'hello world, hi python!'
2 print(text.title())
```

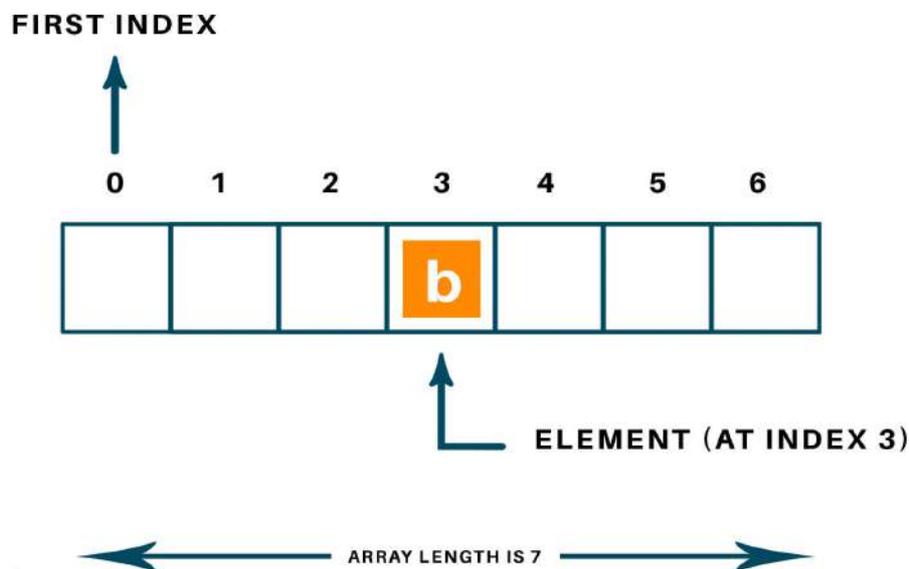
Hello World, Hi Python!

Python Tutorial

Created by Mustafa Germec, PhD

16. Arrays in Python

- Array is a container which can hold a fix number of items and these items should be of the same type.
- Most of the data structures make use of arrays to implement their algorithms.
- Lists can be used to form arrays.
- Following are the important terms to understand the concept of Array.
 - **Element:** Each item stored in an array is called an element.
 - **Index:** Each location of an element in an array has a numerical index, which is used to identify the element.



- Array index begins with 0.
- Each element in the array can be accessed with its index number.
- The length of the array describes the capacity to store the elements.
- Basic array operations are **Traverse, Insertion, Deletion, Search, and Update**.

Creating an array

You should import the module name **'array'** as follows:

- **import array** or **from array import (*)**.
- **(*)** means that it covers all features of the array.

In [4]:

```
1 # Import the module
2 import array as arr
3 from array import *
```

In [5]:

```
1 # To access more information regarding array, you can execute the following commands
2 help(arr)
```

Help on built-in module array:

NAME

array

DESCRIPTION

This module defines an object type which can efficiently represent an array of basic values: characters, integers, floating point numbers. Arrays are sequence types and behave very much like lists, except that the type of objects stored in them is constrained.

CLASSES

builtins.object
array

```
ArrayType = class array(builtins.object)
| array(typecode [, initializer]) -> array
|
| Return a new array whose items are restricted by typecode, and
| initialized from the optional initializer value, which must be a list
```

Type code

- Arrays represent basic values and behave very much like lists, except the type of objects stored in them is constrained.
- The type is specified at object creation time by using a type code, which is a single character.
- The following type codes are defined:

Type code	C Type	Python Type	Minimum size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'q'	signed long long	int	8
'Q'	unsigned long long	int	8
'f'	float	float	4
'd'	double	float	8

In [6]:

```

1 special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
2 for i in special_nums:
3     print(i)

```

```

0.577
1.618
2.718
3.14
6.0
37.0
1729.0

```

Accessing

In [7]:

```

1 special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
2 print(f'First element of numbers is {special_nums[0]}, called euler constant.')
3 print(f'Second element of numbers is {special_nums[1]}, called golden_ratio.')
4 print(f'Last element of numbers is {special_nums[-1]}, called Ramanujan-Hardy number.')

```

```

First element of numbers is 0.577, called euler constant.
Second element of numbers is 1.618, called golden_ratio.
Last element of numbers is 1729.0, called Ramanujan-Hardy number.

```

Changing or Updating

In [8]:

```
1 nums = arr.array('i', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34])
2
3 # Changing the first element of the array
4 nums[0] = 55
5 print(nums)
6
7 # Changing 2nd to 4th elements of the array
8 nums[1:4] = arr.array('i', [89, 144, 233, 377])
9 print(nums)
```

```
array('i', [55, 1, 1, 2, 3, 5, 8, 13, 21, 34])
```

```
array('i', [55, 89, 144, 233, 377, 3, 5, 8, 13, 21, 34])
```

Deleting

In [9]:

```
1 nums = arr.array('i', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34])
2
3 # Deleting the first element of the array
4 del nums[0]
5 print(nums)
6
7 # Deleting the 2nd to 4th elements of the array
8 del nums[1:4]
9 print(nums)
```

```
array('i', [1, 1, 2, 3, 5, 8, 13, 21, 34])
```

```
array('i', [1, 5, 8, 13, 21, 34])
```

Length of the array

In [10]:

```
1 special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
2 print(f'The length of the array is {len(special_nums)}.')
```

The length of the array is 7.

Concatenation

In [11]:

```
1 special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
2 fibonacci_nums = arr.array('d', [1, 1, 2, 3, 5, 8, 13, 21, 34])
3 special_fibonacci_nums = arr.array('d')
4 special_fibonacci_nums = special_nums + fibonacci_nums
5 print(f'The new array called special_fibonacci_nums is {special_fibonacci_nums}.')
```

```
The new array called special_fibonacci_nums is array('d', [0.577, 1.618, 2.718, 3.14, 6.0, 37.0, 1729.0, 1.
```

```
0, 1.0, 2.0, 3.0, 5.0, 8.0, 13.0, 21.0, 34.0]).
```

Creating ID arrays

In [12]:

```
1 mult = 10
2 one_array = [1]*mult
3 print(one_array)
```

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

In [13]:

```
1 mult = 10
2 nums_array = [i for i in range(mult)]
3 print(nums_array)
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Addition with the functions insert() and append()

In [14]:

```
1 # Using the function 'insert()'
2 fibonacci_nums = arr.array('i', [1, 1, 2, 3, 5, 8, 13, 21, 34])
3 print('Before any additon into fibonacci numbers')
4 for i in fibonacci_nums:
5     print(i, end = ' ')
6 print()
7 print('After an element additon into fibonacci numbers')
8 added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
9 fibonacci_nums.insert(9, added_num)
10 for i in fibonacci_nums:
11     print(i, end = ' ')
12 print()
13 print('After an element additon into fibonacci numbers')
14 added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
15 fibonacci_nums.insert(10, added_num)
16 for i in fibonacci_nums:
17     print(i, end = ' ')

```

Before any additon into fibonacci numbers

1 1 2 3 5 8 13 21 34

After an element additon into fibonacci numbers

1 1 2 3 5 8 13 21 34 55

After an element additon into fibonacci numbers

1 1 2 3 5 8 13 21 34 55 89

In [15]:

```
1 # Using the function 'append()'
2 fibonacci_nums = arr.array('i', [1, 1, 2, 3, 5, 8, 13, 21, 34])
3 print('Before any additon into fibonacci numbers')
4 for i in fibonacci_nums:
5     print(i, end = ' ')
6 print()
7 print('After an element additon into fibonacci numbers')
8 added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
9 fibonacci_nums.append(added_num)
10 for i in fibonacci_nums:
11     print(i, end = ' ')
12 print()
13 print('After an element additon into fibonacci numbers')
14 added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
15 fibonacci_nums.append(added_num)
16 for i in fibonacci_nums:
17     print(i, end = ' ')
```

Before any additon into fibonacci numbers

1 1 2 3 5 8 13 21 34

After an element additon into fibonacci numbers

1 1 2 3 5 8 13 21 34 55

After an element additon into fibonacci numbers

1 1 2 3 5 8 13 21 34 55 89

Removing with the function remove() and pop()

In [16]:

```
1 # Using the function 'remove()'
2 special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
3 print('Before removing an element from the array')
4 for i in special_nums:
5     print(i, end = ' ')
6 print()
7 print('After removing an element from the array')
8 special_nums.remove(0.577)
9 for i in special_nums:
10     print(i, end=' ')
11 print()
12 print('After removing one more element from the array')
13 special_nums.remove(special_nums[0]) # We can make this using indexing
14 for i in special_nums:
15     print(i, end=' ')
```

Before removing an element from the array

0.577 1.618 2.718 3.14 6.0 37.0 1729.0

After removing an element from the array

1.618 2.718 3.14 6.0 37.0 1729.0

After removing one more element from the array

2.718 3.14 6.0 37.0 1729.0

In [177]:

```
1 nlis = []
2 nlis.append(math.log10(90))
3 nlis.append(math.log10(1))
4 nlis.append(math.log10(math.e))
5 nlis.append(math.log10(math.pi))
6 nlis.append(math.log10(math.tau))
7 nlis.append(math.log10(math.inf))
8 nlis.append(math.log10(math.nan))
9 print(nlis)
```

[1.954242509439325, 0.0, 0.4342944819032518, 0.49714987269413385, 0.798179868358115, inf, nan]

log1p() function

- Return the natural logarithm of 1+x (base e).

In [179]:

```
1 nlis = []
2 nlis.append(math.log1p(90))
3 nlis.append(math.log1p(1))
4 nlis.append(math.log1p(math.e))
5 nlis.append(math.log1p(math.pi))
6 nlis.append(math.log1p(math.tau))
7 nlis.append(math.log1p(math.inf))
8 nlis.append(math.log1p(math.nan))
9 print(nlis)
```

[4.51085950651685, 0.6931471805599453, 1.3132616875182228, 1.4210804127942926, 1.9855683087099187, inf, nan]

log2() function

- Return the base 2 logarithm of x.

In [183]:

```
1 nlis = []
2 nlis.append(math.log2(90))
3 nlis.append(math.log2(2))
4 nlis.append(math.log2(1))
5 nlis.append(math.log2(math.e))
6 nlis.append(math.log2(math.pi))
7 nlis.append(math.log2(math.tau))
8 nlis.append(math.log2(math.inf))
9 nlis.append(math.log2(math.nan))
10 print(nlis)
```

[6.491853096329675, 1.0, 0.0, 1.4426950408889634, 1.6514961294723187, 2.651496129472319, inf, nan]

modf() function

- It returns the fractional and integer parts of the certain number. Both the outputs carry the sign of x and are of type float.

In [29]:

```
1 print(math.modf(math.pi))
2 print(math.modf(math.e))
3 print(math.modf(1.618))
```

```
(0.14159265358979312, 3.0)
(0.7182818284590451, 2.0)
(0.6180000000000001, 1.0)
```

nextafter() function

- Return the next floating-point value after x towards y.
- if x is equal to y then y is returned.

In [191]:

```
1 nlis = []
2 nlis.append(math.nextafter(3.14, 90))
3 nlis.append(math.nextafter(6, 2.718))
4 nlis.append(math.nextafter(3, math.e))
5 nlis.append(math.nextafter(28, math.inf))
6 nlis.append(math.nextafter(1.618, math.nan))
7 nlis.append(math.nextafter(1, 1))
8 nlis.append(math.nextafter(0, 0))
9 print(nlis)
```

```
[3.1400000000000006, 5.999999999999999, 2.9999999999999996, 28.000000000000004, nan, 1.0, 0.0]
```

perm() function

- Returns the number of ways to choose k items from n items with order and without repetition.

In [31]:

```
1 print(math.perm(6, 2))
2 print(math.perm(6, 6))
```

```
30
720
```

pow() function

- Returns the value of x to the power of y.

In [34]:

```
1 print(math.pow(10, 2))
2 print(math.pow(math.pi, math.e))
```

100.0

22.45915771836104

prod() function

- Returns the product of all the elements in an iterable

In [32]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2 print(math.prod(special_nums))
```

85632659.07026622

radians() function

- Convert angle x from degrees to radians.

In [193]:

```
1 nlis = []
2 nlis.append(math.radians(0))
3 nlis.append(math.radians(30))
4 nlis.append(math.radians(45))
5 nlis.append(math.radians(60))
6 nlis.append(math.radians(90))
7 nlis.append(math.radians(120))
8 nlis.append(math.radians(180))
9 nlis.append(math.radians(270))
10 nlis.append(math.radians(360))
11 print(nlis)
```

```
[0.0, 0.5235987755982988, 0.7853981633974483, 1.0471975511965976, 1.5707963267948966, 2.0943
951023931953, 3.141592653589793, 4.71238898038469, 6.283185307179586]
```

remainder() function

- Difference between x and the closest integer multiple of y.
- Return $x - ny$ where ny is the closest integer multiple of y.
- ReturnIn the case where x is exactly halfway between two multiples of
- y, the nearest even value of n is used. The result is always exact.

In [196]:

```
1 nlis = []
2 nlis.append(math remainder(3.14, 2.718))
3 nlis.append(math remainder(6, 28))
4 nlis.append(math remainder(5, 3))
5 nlis.append(math remainder(1729, 37))
6 print(nlis)
```

```
[0.422000000000000015, 6.0, -1.0, -10.0]
```

sin() function

- Return the sine of x (measured in radians).
- **Note:** To find the sine of degrees, it must first be converted into radians with the **math.radians() method**.

In [204]:

```
1 nlis = []
2 nlis.append(math.sin(math.pi))
3 nlis.append(math.sin(math.pi/2))
4 nlis.append(math.sin(math.e))
5 nlis.append(math.sin(math.nan))
6 nlis.append(math.sin(math.tau))
7 nlis.append(math.sin(30))
8 nlis.append(math.sin(-5))
9 nlis.append(math.sin(37))
10 print(nlis)
```

```
[1.2246467991473532e-16, 1.0, 0.41078129050290885, nan, -2.4492935982947064e-16, -0.988031624
0928618, 0.9589242746631385, -0.6435381333569995]
```

sinh() function

- Return the hyperbolic sine of x.

In [213]:

```
1 nlis = []
2 nlis.append(math.sinh(1))
3 nlis.append(math.sinh(0))
4 nlis.append(math.sinh(-5))
5 nlis.append(math.sinh(math.pi))
6 nlis.append(math.sinh(math.e))
7 nlis.append(math.sinh(math.tau))
8 nlis.append(math.sinh(math.nan))
9 nlis.append(math.sinh(math.inf))
10 print(nlis)
```

```
[1.1752011936438014, 0.0, -74.20321057778875, 11.548739357257746, 7.544137102816975, 267.744
89404101644, nan, inf]
```

sqrt() function

- Return the square root of x .

In [210]:

```
1 nlis = []
2 nlis.append(math.sqrt(1))
3 nlis.append(math.sqrt(0))
4 nlis.append(math.sqrt(37))
5 nlis.append(math.sqrt(math.pi))
6 nlis.append(math.sqrt(math.e))
7 nlis.append(math.sqrt(math.tau))
8 nlis.append(math.sqrt(math.nan))
9 nlis.append(math.sqrt(math.inf))
10 print(nlis)
```

```
[1.0, 0.0, 6.082762530298219, 1.7724538509055159, 1.6487212707001282, 2.5066282746310002, nan, inf]
```

tan() function

- Return the tangent of x (measured in radians).

In [212]:

```
1 nlis = []
2 nlis.append(math.tan(0))
3 nlis.append(math.tan(30))
4 nlis.append(math.tan(45))
5 nlis.append(math.tan(60))
6 nlis.append(math.tan(90))
7 nlis.append(math.tan(120))
8 nlis.append(math.tan(180))
9 nlis.append(math.tan(270))
10 nlis.append(math.tan(360))
11 print(nlis)
```

```
[0.0, -6.405331196646276, 1.6197751905438615, 0.320040389379563, -1.995200412208242, 0.7131230097859091, 1.3386902103511544, -0.17883906379845224, -3.380140413960958]
```

tanh() function

- Return the hyperbolic tangent of x .

In [214]:

```
1 nlis = []
2 nlis.append(math.tanh(1))
3 nlis.append(math.tanh(0))
4 nlis.append(math.tanh(-5))
5 nlis.append(math.tanh(math.pi))
6 nlis.append(math.tanh(math.e))
7 nlis.append(math.tanh(math.tau))
8 nlis.append(math.tanh(math.nan))
9 nlis.append(math.tanh(math.inf))
10 print(nlis)
```

```
[0.7615941559557649, 0.0, -0.9999092042625951, 0.99627207622075, 0.9913289158005998, 0.99999
30253396107, nan, 1.0]
```

trunc() function

- Truncates the Real x to the nearest Integral toward 0.
- Returns the truncated integer parts of different numbers

In [218]:

```
1 nlis = []
2 nlis.append(math.trunc(1))
3 nlis.append(math.trunc(0))
4 nlis.append(math.trunc(-5))
5 nlis.append(math.trunc(0.577))
6 nlis.append(math.trunc(1.618))
7 nlis.append(math.trunc(math.pi))
8 nlis.append(math.trunc(math.e))
9 nlis.append(math.trunc(math.tau))
10 print(nlis)
```

```
[1, 0, -5, 0, 1, 3, 2, 6]
```

ulp() function

- Return the value of the least significant bit of the float x .

In [224]:

```
1 import sys
2 nlis = []
3 nlis.append(math.ulp(1))
4 nlis.append(math.ulp(0))
5 nlis.append(math.ulp(-5))
6 nlis.append(math.ulp(0.577))
7 nlis.append(math.ulp(1.618))
8 nlis.append(math.ulp(math.pi))
9 nlis.append(math.ulp(math.e))
10 nlis.append(math.ulp(math.tau))
11 nlis.append(math.ulp(math.nan))
12 nlis.append(math.ulp(math.inf))
13 nlis.append(math.ulp(-math.inf))
14 nlis.append(math.ulp(float('nan')))
15 nlis.append(math.ulp(float('inf')))
16 x = sys.float_info.max
17 nlis.append(math.ulp(x))
18 print(nlis)
```

```
[2.220446049250313e-16, 5e-324, 8.881784197001252e-16, 1.1102230246251565e-16, 2.220446049250313e-16, 4.440892098500626e-16, 4.440892098500626e-16, 8.881784197001252e-16, nan, inf, inf, nan, inf, 1.99584030953472e+292]
```

Python Tutorial

Created by Mustafa Germec, PhD

19. List Comprehension in Python

- **List comprehension** in Python is an easy and compact syntax for creating a list from a string or another list.
- It is a very concise way to create a new list by performing an operation on each item in the existing list.
- **List comprehension** is considerably **faster** than processing a list using the for loop.



Examples

In [3]:

```
1 import math
2 from math import *
```

In [24]:

```
1 # Using for loop
2 cubic_nums = []
3 for i in range(5):
4     i**=3
5     cubic_nums.append(i)
6 print(cubic_nums)
7
8 # Using list comprehension
9 cubic_nums = [i**3 for i in range(5)]
10 print(cubic_nums)
11
12 # Using list comprehension
13 cubic_nums = [math.pow(i, 3) for i in range(5)]
14 print(cubic_nums)
```

```
[0, 1, 8, 27, 64]
```

```
[0, 1, 8, 27, 64]
```

```
[0.0, 1.0, 8.0, 27.0, 64.0]
```

In [90]:

```
1 # Using for loop
2 even_numbers = []
3 for i in range(21):
4     if i%2 == 0:
5         even_numbers.append(i)
6 print(even_numbers)
7
8 # Using list comprehension
9 even_numbers = [i for i in range(21) if i%2==0]
10 print(even_numbers)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Example: The number of insects in a lab doubles in size every month. Take the initial number of insects as input and output a list, showing the number of insects for each of the next 12 months, starting with 0, which is the initial value. So the resulting list should contain 12 items, each showing the number of insects at the beginning of that month.

In [48]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
2
3 # Using for loop
4 new_lis = []
5 for i in special_nums:
6     if i < 5:
7         new_lis.append(i)
8 print(new_lis)
9
10 # Using list comprehension
11 new_lis = [i for i in special_nums if i < 5]
12 print(new_lis)
```

```
[0.577, 1.618, 2.718, 3.14]
```

```
[0.577, 1.618, 2.718, 3.14]
```

In [51]:

```
1 languages = ['Java', 'JavaScript', 'C', 'C++', 'PHP']
2 # Using for loop
3 lang_lis = []
4 for i in languages:
5     i = 'Python'
6     lang_lis.append(i)
7 print(lang_lis)
8
9 #Using list comprehension
10 lang_lis = ['Python' for i in languages]
11 print(lang_lis)
```

```
['Python', 'Python', 'Python', 'Python', 'Python']
```

```
['Python', 'Python', 'Python', 'Python', 'Python']
```

In [53]:

```
1 languages = ['Java', 'JavaScript', 'C', 'C++', 'PHP']
2 # Using for loop
3 lang_lis = []
4 for i in languages:
5     if i != 'Java':
6         lang_lis.append(i)
7     else:
8         lang_lis.append('Python')
9 print(lang_lis)
10
11
12 #Using list comprehension
13 lang_lis = [i if i != 'Java' else 'Python' for i in languages]
14 print(lang_lis)
```

```
['Python', 'JavaScript', 'C', 'C++', 'PHP']
```

```
['Python', 'JavaScript', 'C', 'C++', 'PHP']
```

In [56]:

```
1 languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']
2 # Using for loop
3 lang_lis = []
4 for i in languages:
5     i = i.upper()
6     lang_lis.append(i)
7 print(lang_lis)
8
9 #Using list comprehension
10 lang_lis = [i.upper() for i in languages]
11 print(lang_lis)
```

```
['PYTHON', 'JAVA', 'JAVASCRIPT', 'C', 'C++', 'PHP']
```

```
['PYTHON', 'JAVA', 'JAVASCRIPT', 'C', 'C++', 'PHP']
```

In [59]:

```
1 # Using for loop
2 python = []
3 for i in 'Python':
4     python.append(i)
5 print(python)
6
7 # Using list comprehension
8 python = [i for i in 'Python']
9 print(python)
10
11 # Using lambda function
12 python = list(map(lambda i: i, 'Python'))
13 print(python)
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

```
['P', 'y', 't', 'h', 'o', 'n']
```


In [89]:

```
1 # Transpose of 2D matrix
2 matrix = [[0.577, 1.618, 0],
3           [2.718, 3.14, 1],
4           [6, 28, 28]]
5 transpose_matrix = [[i[j] for i in matrix] for j in range(len(matrix))]
6 print(transpose_matrix)
```

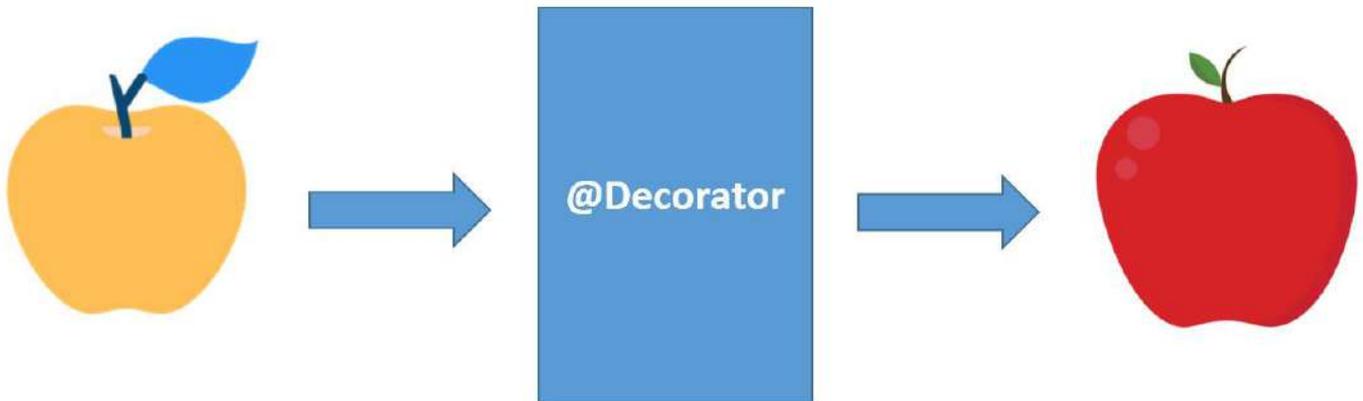
```
[[0.577, 2.718, 6], [1.618, 3.14, 28], [0, 1, 28]]
```

Python Tutorial

Created by Mustafa Germec, PhD

20. Decorators in Python

- **Decorators** provide a simple syntax for calling **higher-order functions**.
- By definition, a **decorator** is a function that takes another function and extends the behavior of the latter function without explicitly modifying it.
- A **decorator** in Python is a function that takes another function as its argument, and returns yet another function.
- **Decorators** can be extremely useful as they allow the extension of an existing function, without any modification to the original function source code.
- In fact, there are **two types of decorators** in Python including **class decorators and function decorators**.
- In application, **decorators** are majorly used in creating middle layer in the backend, it performs task like token authentication, validation, image compression and many more.



Syntax for Decorator

In []:

```
1 """
2 @hello_decorator
3 def hi_decorator():
4     print("Hello")
5 """
6
7 ""
8 Above code is equal to -
9
10 def hi_decorator():
11     print("Hello")
12
13 hi_decorator = hello_decorator(hi_decorator)
14 ""
```

In [5]:

```
1 # Import libraries
2 import decorator
3 from decorator import *
4 import functools
5 import math
```

In [26]:

```
1 help(decorator)
```

Help on function decorator in module decorator:

```
decorator(caller, _func=None, kwsyntax=False)
decorator(caller) converts a caller function into a decorator
```

Functions

In [27]:

```
1 # Define a function
2 """
3 In the following function, when the code was executed, it yeilds the outputs for both functions.
4 The function new_text() alluded to the function mytext() and behave as function.
5 """
6 def mytext(text):
7     print(text)
8
9 mytext('Python is a programming language.')
10 new_text = mytext
11 new_text('Hell, Python!')
```

```
Python is a programming language.
Hell, Python!
```

In [1]:

```
1 def multiplication(num):
2     return num * num
3
4 mult = multiplication
5 mult(3.14)
```

Out[1]:

9.8596

Nested/Inner Function

In [28]:

```
1 # Define a function
2 """
3 In the following function, it is nonsignificant how the child functions are announced.
4 The implementation of the child function does influence on the output.
5 These child functions are topically linked with the function mytext(), therefore they can not be called individually.
6 """
7 def mytext():
8     print('Python is a programming language.')
9     def new_text():
10        print('Hello, Python!')
11    def message():
12        print('Hi, World!')
13
14    new_text()
15    message()
16 mytext()
17
```

Python is a programming language.
Hello, Python!
Hi, World!

In [3]:

```
1 # Define a function
2 """
3 In the following example, the function text() is nesred into the function message().
4 It will return each time when the function tex() is called.
5 """
6 def message():
7     def text():
8         print('Python is a programming language.')
9         return text
10
11 new_message = message()
12 new_message()
```

Python is a programming language.

In [4]:

```
1 def function(num):
2     def mult(num):
3         return num*num
4
5     output = mult(num)
6     return output
7 mult(3.14)
```

Out[4]:

9.8596

In [13]:

```
1 def msg(text):
2     'Hello, World!'
3     def mail():
4         'Hi, Python!'
5         print(text)
6
7     mail()
8
9 msg('Python is the most popular programming language.')
```

Python is the most popular programming language.

Passing functions

In [29]:

```
1 # Define a function
2 """
3 In this function, the mult() and divide() functions as argument in operator() function are passed.
4 """
5 def mult(x):
6     return x * 3.14
7 def divide(x):
8     return x/3.14
9 def operator(function, x):
10    number = function(x)
11    return number
12
13 print(operator(mult, 2.718))
14 print(operator(divide, 1.618))
```

8.53452

0.5152866242038217

In [7]:

```

1 def addition(num):
2     return num + math.pi
3
4 def called_function(func):
5     added_number = math.e
6     return func(added_number)
7
8 called_function(addition)

```

Out[7]:

5.859874482048838

In [111]:

```

1 def decorator_one(function):
2     def inner():
3         num = function()
4         return num * (num**num)
5     return inner
6
7 def decorator_two(function):
8     def inner():
9         num = function()
10        return (num**num)/num
11    return inner
12
13 @decorator_one
14 @decorator_two
15 def number():
16     return 4
17
18 print(number())
19
20 # The above decorator returns the following code
21 x = pow(4, 4)/4
22 print(x*(x**x))

```

2.5217283965692467e+117

2.5217283965692467e+117

Functions reverting other functions

In [11]:

```

1 def msg_func():
2     def text():
3         return "Python is a programming language."
4     return text
5 msg = msg_func()
6 print(msg())

```

Python is a programming language.

Decorating functions

In [8]:

```
1 # Define a decorating function
2 """
3 In the following example, the function outer_addition that is some voluminous is decorated.
4 """
5 def addition(a, b):
6     print(a+b)
7 def outer_addition(func):
8     def inner(a, b):
9         if a < b:
10             a, b = b, a
11         return func(a, b)
12     return inner
13
14 result = outer_addition(addition)
15 result(math.pi, math.e)
```

5.859874482048838

In [9]:

```
1 """
2 Rather than above function, Python ensures to employ decorator in easy way with the symbol @ called 'pie' syntax, as
3 """
4 def outer_addition(function):
5     def inner(a, b):
6         if a < b:
7             a, b = b, a
8         return function(a, b)
9     return inner
10
11 @outer_addition    # Syntax of decorator
12 def addition(a, b):
13     print(a+b)
14 result = outer_addition(addition)
15 result(math.pi, math.e)
```

5.859874482048838

In [17]:

```
1 def decorator_text_uppercase(func):
2     def wrapper():
3         function = func()
4         text_uppercase = function.upper()
5         return text_uppercase
6
7     return wrapper
8
9 # Using a function
10 def text():
11     return 'Python is the most popular programming language.'
12
13 decorated_result = decorator_text_uppercase(text)
14 print(decorated_result())
15
16 # Using a decorator
17 @decorator_text_uppercase
18 def text():
19     return 'Python is the most popular programming language.'
20
21 print(text())
```

PYTHON IS THE MOST POPULAR PROGRAMMING LANGUAGE.
PYTHON IS THE MOST POPULAR PROGRAMMING LANGUAGE.

Reprocessing decorator

- The decorator can be reused by recalling that decorator function.

In [37]:

```
1 def do_twice(function):
2     def wrapper_do_twice():
3         function()
4         function()
5     return wrapper_do_twice
6
7 @do_twice
8 def text():
9     print('Python is a programming language.')
10 text()
```

Python is a programming language.
Python is a programming language.

Decorators with Arguments

In [39]:

```
1 def do_twice(function):
2     """
3     The function wrapper_function() can admit any number of argument and pass them on the function.
4     """
5     def wrapper_function(*args, **kwargs):
6         function(*args, **kwargs)
7         function(*args, **kwargs)
8     return wrapper_function
9
10 @do_twice
11 def text(programming_language):
12     print(f'{programming_language} is a programming language.')
13 text('Python')
```

Python is a programming language.

Python is a programming language.

Returning values from decorated function

In [41]:

```
1 @do_twice
2 def returning(programming_language):
3     print('Python is a programming language.')
4     return f'Hello, {programming_language}'
5
6 hello_python = returning('Python')
```

Python is a programming language.

Python is a programming language.

Fancy decorators

- @propertymethod
- @staticmethod
- @classmethod

In [45]:

```
1 class Microorganism:
2     def __init__(self, name, product):
3         self.name = name
4         self.product = product
5     @property
6     def show(self):
7         return self.name + ' produces ' + self.product + ' enzyme'
8
9 organism = Microorganism('Aspergillus niger', 'inulinase')
10 print(f'Microorganism name: {organism.name}')
11 print(f'Microorganism product: {organism.product}')
12 print(f'Message: {organism.show}.')
```

Microorganism name: Aspergillus niger
Microorganism product: inulinase
Message: Aspergillus niger produces inulinase enzyme.

In [46]:

```
1 class Micoorganism:
2     @staticmethod
3     def name():
4         print('Aspergillus niger is a fungus that produces inulinase enzyme.')
5
6 organisms = Micoorganism()
7 organisms.name()
8 Micoorganism.name()
```

Aspergillus niger is a fungus that produces inulinase enzyme.
Aspergillus niger is a fungus that produces inulinase enzyme.

In [97]:

```
1 class Microorganism:
2     def __init__(self, name, product):
3         self.name = name
4         self.product = product
5
6     @classmethod
7     def display(cls):
8         return cls('Aspergillus niger', 'inulinase')
9
10 organism = Microorganism.display()
11 print(f'The fungus {organism.name} produces {organism.product} enzyme.')
12
```

The fungus Aspergillus niger produces inulinase enzyme.

Decorator with arguments

In [49]:

```
1 """
2 In the following example, @iterate refers to a function object that can be called in another function.
3 The @iterate(numbers=4) will return a function which behaves as a decorator.
4 """
5 def iterate(numbers):
6     def decorator_iterate(function):
7         @functools.wraps(function)
8         def wrapper(*args, **kwargs):
9             for _ in range(numbers):
10                 worth = function(*args, **kwargs)
11             return worth
12         return wrapper
13     return decorator_iterate
14
15 @iterate(numbers=4)
16 def function_one(name):
17     print(f'{name}')
18
19 x = function_one('Python')
```

Python

Python

Python

Python

In [21]:

```
1 def arguments(func):
2     def wrapper_arguments(argument_1, argument_2):
3         print(f'The arguments are {argument_1} and {argument_2}.')
4         func(argument_1, argument_2)
5     return wrapper_arguments
6
7
8 @arguments
9 def programing_language(lang_1, lang_2):
10     print(f'My favorite programming languages are {lang_1} and {lang_2}.')
11
12 programing_language("Python", "R")
```

The arguments are Python and R.

My favorite programming languages are Python and R.

Multiple decorators

In [18]:

```
1 def splitted_text(text):
2     def wrapper():
3         function = text()
4         text_splitting = function.split()
5         return text_splitting
6
7     return wrapper
8
9 @splitted_text
10 @decorator_text_uppercase    # Calling other decorator above
11 def text():
12     return 'Python is the most popular programming language.'
13 text()
```

Out[18]:

```
['PYTHON', 'IS', 'THE', 'MOST', 'POPULAR', 'PROGRAMMING', 'LANGUAGE.']
```

Arbitrary arguments

In [43]:

```

1 def arbitrary_argument(func):
2     def wrapper(*args,**kwargs):
3         print(f'These are positional arguments {args}.')
4         print(f'These are keyword arguments {kwargs}.')
5         func(*args)
6     return wrapper
7
8 """1. Without arguments decorator"""
9 print(__doc__)
10 @arbitrary_argument
11 def without_argument():
12     print("There is no argument in this decorator.")
13
14 without_argument()
15
16 """2. With positional arguments decorator"""
17 print(__doc__)
18 @arbitrary_argument
19 def with_positional_argument(x1, x2, x3, x4, x5, x6):
20     print(x1, x2, x3, x4, x5, x6)
21
22 with_positional_argument(math.inf, math.tau, math.pi, math.e, math.nan, -math.inf)
23
24 """3. With keyword arguments decorator"""
25 print(__doc__)
26 @arbitrary_argument
27 def with_keyword_argument():
28     print("Python and R are my favorite programming languages and keyword arguments.")
29
30 with_keyword_argument(language_1="Python", language_2="R")

```

1. Without arguments decorator

These are positional arguments ().

These are keyword arguments {}.

There is no argument in this decorator.

2. With positional arguments decorator

These are positional arguments (inf, 6.283185307179586, 3.141592653589793, 2.718281828459045, nan, -inf).

These are keyword arguments {}.

inf 6.283185307179586 3.141592653589793 2.718281828459045 nan -inf

3. With keyword arguments decorator

These are positional arguments ().

These are keyword arguments {'language_1': 'Python', 'language_2': 'R'}.

Python and R are my favorite programming languages and keyword arguments.

Debugging decorators

In [69]:

```
1 def capitalize_dec(function):
2     @functools.wraps(function)
3     def wrapper():
4         return function().capitalize()
5     return wrapper
6
7 @capitalize_dec
8 def message():
9     "Python is the most popular programming language."
10    return 'PYTHON IS THE MOST POPULAR PROGRAMMING LANGUAGE. '
11
12 print(message())
13 print()
14 print(message.__name__)
15 print(message.__doc__)
```

Python is the most popular programming language.

message

Python is the most popular programming language.

Preserving decorators

In [85]:

```
1 def preserved_decorator(function):
2     def wrapper():
3         print('Before calling the function, this is printed.')
4         function()
5         print('After calling the function, this is printed.')
6     return wrapper
7
8 @preserved_decorator
9 def message():
10     """This function prints the message when it is called."""
11     print('Python is the most popular programming language.')
12
13 message()
14 print(message.__name__)
15 print(message.__doc__)
16 print(message.__class__)
17 print(message.__module__)
18 print(message.__code__)
19 print(message.__closure__)
20 print(message.__annotations__)
21 print(message.__dir__)
22 print(message.__format__)
```

Before calling the function, this is printed.

Python is the most popular programming language.

After calling the function, this is printed.

wrapper

None

<class 'function'>

__main__

<code object wrapper at 0x0000029986F96970, file "C:\Users\test\AppData\Local\Temp\ipykernel_1180/3788198392.py", line 2>

(<cell at 0x0000029986311840: function object at 0x0000029981A03F40>,)
{}

{}

<built-in method __dir__ of function object at 0x0000029986273250>

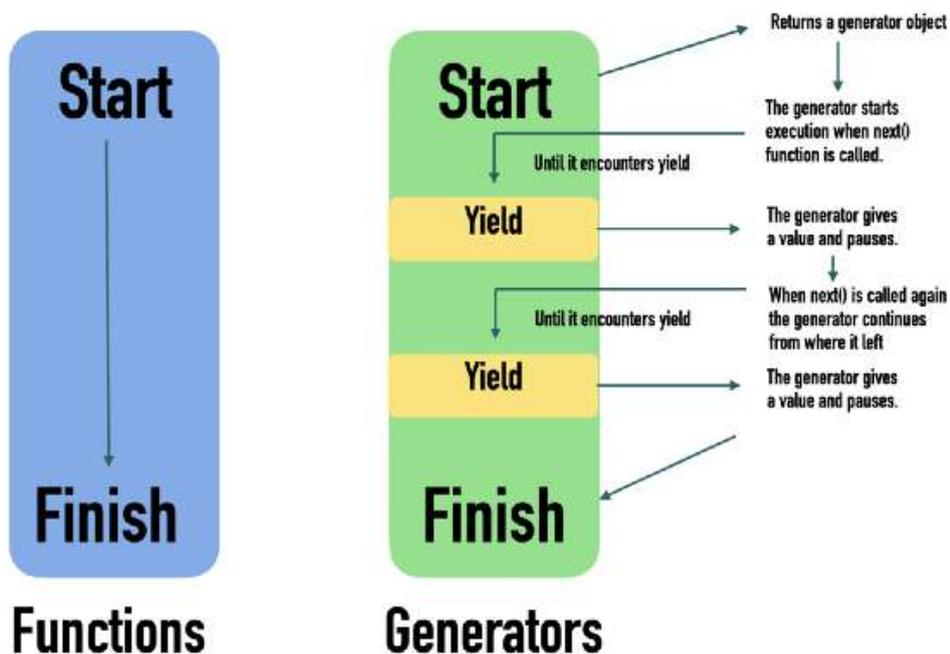
<built-in method __format__ of function object at 0x0000029986273250>

Python Tutorial

Created by Mustafa Germec, PhD

21. Generators in Python

- Python generators are the functions that return the traversal object and a simple way of creating iterators.
- It traverses the entire items at once.
- The generator can also be an expression in which syntax is similar to the list comprehension in Python.
- There is a lot of complexity in creating iteration in Python, it is required to implement `iter()` and `next()` methods to keep track of internal states.
- It is a lengthy process to create iterators.
- That is why the generator plays a significant role in simplifying this process.
- If there is no value found in iteration, it raises **StopIteration** exception.
- It is quite simple to create a generator in Python.
- It is similar to the normal function defined by the `def` keyword and employs a `yield` keyword instead of `return`.
- If the body of any function includes a `yield` statement, it automatically becomes a **generator function**.
- The `yield` keyword is responsible to control the flow of the generator function.
- It pauses the function execution by saving all states and yielded to the caller.
- Later it resumes execution when a successive function is called.
- The `return` keyword returns a value and terminates the whole function and only one return statement can be employed in the function.



In [22]:

```
1 def function():
2     for i in range(10):
3         if i%2==0:
4             yield i
5
6 nlis = []
7 for i in function():
8     nlis.append(i)
9 print(nlis)
```

[0, 2, 4, 6, 8]

In [26]:

```
1 def func():
2     for i in range(25):
3         if i%4==0:
4             yield i
5
6 num_lis = []
7 for i in func():
8     num_lis.append(i)
9 print(num_lis)
```

[0, 4, 8, 12, 16, 20, 24]

In [2]:

```
1 def message():
2     msg_one = 'Hello, World!'
3     yield msg_one
4
5     msg_two = 'Hi, Python!'
6     yield msg_two
7
8     msg_three = 'Python is the most popular programming language.'
9     yield msg_three
10
11 result = message()
12 print(next(result))
13 print(next(result))
14 print(next(result))
```

Hello, World!

Hi, Python!

Python is the most popular programming language.

In [4]:

```
1 """
2 In the following example, the list comprehension will return the list of cube of elements.
3 Whereas the generator expression will return the reference of the calculated value.
4 Rather than this application, the ^function 'next()' can be used on the generator object.
5 """
6 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 37, 1729]
7
8 list_comp = [i*3 for i in special_nums] # This is a list comprehension.
9 generator_exp = (i*3 for i in special_nums) # This is a generator expression.
10
11 print(list_comp)
12 print(generator_exp)
```

```
[1.7309999999999999, 4.854, 8.154, 9.42, 18, 111, 5187]
<generator object <genexpr> at 0x000002572F0E1230>
```

In [8]:

```
1 special_nums = [0.577, 1.618, 2.718, 3.14, 6, 37, 1729]
2
3 generator_exp = (i*3 for i in special_nums) # This is a generator expression.
4
5 nums_list = []
6 nums_list.append(next(generator_exp))
7 nums_list.append(next(generator_exp))
8 nums_list.append(next(generator_exp))
9 nums_list.append(next(generator_exp))
10 nums_list.append(next(generator_exp))
11 nums_list.append(next(generator_exp))
12 nums_list.append(next(generator_exp))
13 print(nums_list)
```

```
[1.7309999999999999, 4.854, 8.154, 9.42, 18, 111, 5187]
```

In [12]:

```
1 def mult_table(n):
2     for i in range(0, 11):
3         yield n*i
4         i+=1
5
6 mult_table_list = []
7 for i in mult_table(20):
8     mult_table_list.append(i)
9 print(mult_table_list)
```

```
[0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200]
```

In [17]:

```
1 import sys
2
3 # List comprehension
4 cubic_nums_lc = [i**3 for i in range(1500)]
5 print(f'Memory in bytes with list comprehension is {sys.getsizeof(cubic_nums_lc)}.')
6
7 # Generator expression of the same conditions
8 cubic_nums_gc = (i**3 for i in range(1500))
9 print(f'Memory in bytes with generator expression is {sys.getsizeof(cubic_nums_gc)}.')
```

Memory in bytes with list comprehension is 12728.

Memory in bytes with generator expression is 104.

- You can find more information by executing the following command
- **help(sys)**

The following generator produces infinite numbers.

In []:

```
1 def infinite():
2     count = 0
3     while True:
4         yield count
5         count = count + 1
6
7 for i in infinite():
8     print(i)
```

In [29]:

```
1 def generator(a):
2     for i in range(a):
3         yield i
4
5 gen = generator(6)
6 print(next(gen))
7 print(next(gen))
8 print(next(gen))
9 print(next(gen))
10 print(next(gen))
11 print(next(gen))
12 print(next(gen))
```

```
0
1
2
3
4
5
```

StopIteration Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_20708\3124683773.py in <module>
 10 print(next(gen))
 11 print(next(gen))
--> 12 print(next(gen))

StopIteration:

In [41]:

```
1 def square_number(num):
2     for i in range(num):
3         yield i**i
4
5 generator = square_number(6)
6
7 # Using 'while' loop
8 while True:
9     try:
10        print(f'The number using while loop is {next(generator)}.')
11    except StopIteration:
12        break
13
14 # Using 'for' loop
15 nlis = []
16 for square in square_number(6):
17     nlis.append(square)
18 print(f'The numbers using for loop are {nlis}.')
19
20 # Using generator comprehension
21 square = (i**i for i in range(6))
22 square_list = []
23 square_list.append(next(square))
24 square_list.append(next(square))
25 square_list.append(next(square))
26 square_list.append(next(square))
27 square_list.append(next(square))
28 square_list.append(next(square))
29 print(f'The numbers using generator comprehension are {square_list}.')
```

The number using while loop is 1.

The number using while loop is 1.

The number using while loop is 4.

The number using while loop is 27.

The number using while loop is 256.

The number using while loop is 3125.

The numbers using for loop are [1, 1, 4, 27, 256, 3125].

The numbers using generator comprehension are [1, 1, 4, 27, 256, 3125].

In [42]:

```
1 import math
2 sum(i**i for i in range(6))
```

Out[42]:

3414

In [46]:

```
1 def fibonacci(numbers):
2     a, b = 0, 1
3     for _ in range(numbers):
4         a, b = b, a+b
5         yield a
6
7 def square(numbers):
8     for i in numbers:
9         yield i**2
10
11 print(sum(square(fibonacci(25))))
```

9107509825