# Python Tutorial (Codes)

**Mustafa GERMEC, PhD**

# TABLE OF CONTENTS

## PYTHON TUTORIAL

*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]:

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

```

```
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]:

```
# The Python help function is used to display the documentation of modules, functions, classes, keywords, etc.
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]:

```
# This is a comment, and to write a comment, '#' symbol is used.
print('Hello World!')      # This line prints a string.

# Print 'Hello'
print('Hello')
```

```
Hello World!
Hello
```

### Errors

In [13]:

```
# Print string as error message
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]:

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

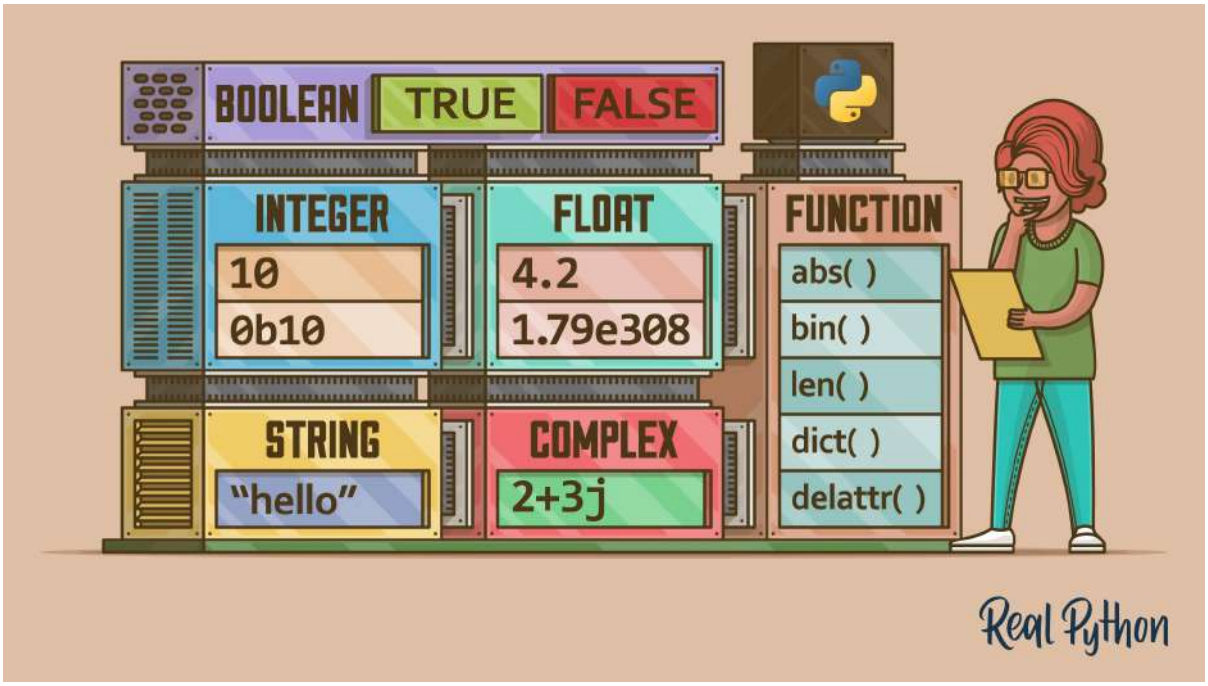
```
  File "C:\Users\test\AppData\Local\Temp/ipykernel_13804/974508531.py", line 2
    print('Hello, World!)
          ^
SyntaxError: unterminated string literal (detected at line 2)
```

In [15]:

```
# Print both string and error to see the running order
print('This string is printed')
frint('This gives an error message')
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]:

```
# String
print("Hello, World!")
# Integer
print(12)
# Float
print(3.14)
# Boolean
print(True)
print(False)
print(bool(1))    # Output = True
print(bool(0))    # Output = False

```

```
Hello, World!
12
3.14
True
False
True
False
```

### type() function

In [29]:

```
# String
print(type('Hello, World!'))

# Integer
print(type(15))
print(type(-24))
print(type(0))
print(type(1))

# Float
print(type(3.14))
print(type(0.5))
print(type(1.0))
print(type(-5.0))

# Boolean
print(type(True))
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]:

```
# to obtain the information about 'interger' and 'float'
print(sys.int_info)
print()                 # to add a space between two outputs, use 'print()' function
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 abject type to another object type

In [35]:

```
# Let's convert the integer number 6 to a string and a float

number = 6

print(str(number))
print(float(number))
print(type(number))
print(type(str(number)))
print(type(float(number)))
str(number)
```

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

Out[35]:

```
'6'
```

In [37]:

```
# Let's conver the float number 3.14 to a string and an integer

number = 3.14

print(str(number))
print(int(number))
print(type(number))
print(type(str(number)))
print(type(int(number)))
str(number)
```

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

Out[37]:

```
'3.14'
```

In [42]:

```
# Let's convert the booleans to an integer,  a float, and a string

bool_1 = True
bool_2 = False

print(int(bool_1))
print(int(bool_2))
print(float(bool_1))
print(float(bool_2))
print(str(bool_1))
print(str(bool_2))
print(bool(1))
print(bool(0))
```

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

In [46]:

```
# Let's find the data types of 9/3 and 9//4

print(9/3)
print(9//4)
print(type(9/3))
print(type(9//4))
```

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

### Experesion and variables

In [47]:

```
# Addition

x = 56+65+89+45+78.5+98.2
print(x)
print(type(x))
```

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

In [48]:

```
# Substraction

x = 85-52-21-8
print(x)
print(type(x))
```

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

In [49]:

```
# Multiplication

x = 8*74
print(x)
print(type(x))
```

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

In [50]:

```
# Division

x = 125/24
print(x)
print(type(x))
```

```
5.208333333333333
<class 'float'>
```

In [51]:

```
# Floor division

x = 125//24
print(x)
print(type(x))
```

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

In [52]:

```
# Modulus

x = 125%24
print(x)
print(type(x))
```

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

In [54]:

```
# Exponentiation

x = 2**3
print(x)
print(type(x))
```

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

In [56]:

```
# An example: Let's calculate how many minutes there are in 20 hours?

one_hour = 60      # 60 minutes
hour = 20
minutes = one_hour *hour
print(minutes)
print(type(minutes))

# An example: Let's calculate how many hours there are in 348 minutes?

minutes = 348
one_hour = 60
hours = 348/60
print(hours)
print(type(hours))
```

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

In [57]:

```
# Mathematica expression
x = 45+3*89
y = (45+3)*89
print(x)
print(y)
print(x+y)
print(x-y)
print(x*y)
print(x/y)
print(x**y)
print(x//y)
print(x%y)
```

```
312
4272
4584
-3960
1332864
0.07303370786516854
106764199167287649605554376373081784961189430306931493889556878541263403954002216688428743890341298063062142643611547988366237942127177343103591136201873077048553130787246373784413835009801652141537511130496428252345316433301059252139523910338594414308819431610621847043225489424826149872487789309094682282558124209932422054457355942893935706933289840196191187747301112830107448513231858429992761218679164101636444032930435771562516453083564435414559235582600151873226528728740867781322733341290526168852400525662403862366229423780827737199759399891266789683171279214118065400092433700677527805247487272637725301042917923096127461019970997201882165678942340635917406021261129472798657195977765495201179425063701798535808090821660144758848122559902003139072857327121828979686902128532381362533527097401887285523369419688233628863002122383440451166119429893245226499915609903372771385548085435537115059973855787871297757754927143334381337974992965756110903298883558058521609264061222316457091352551267002967383462418697013273188501
```

## Variables

In [58]:

```
# Store the value 89 into the variabe 'number'

number = 90
print(number)
print(type(number))
```

```
90
<class 'int'>
```

In [62]:

```
x = 25
y = 87
z = 5*x - 2*y
print(z)

t = z/7
print(t)

z = z/14
print(z)
```

```
-49
-7.0
-3.5
```

In [68]:

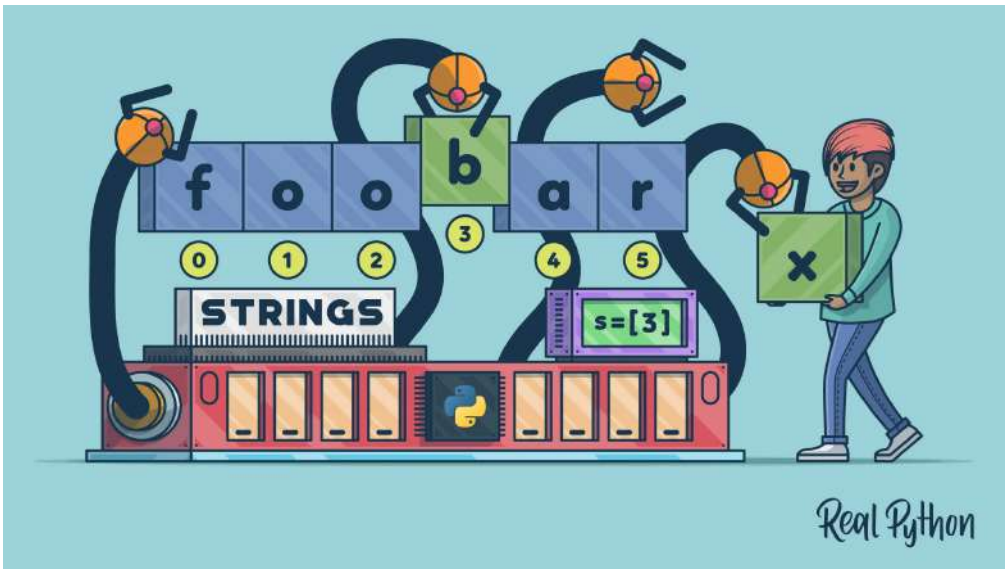
```
x, y, z = 8, 4, 2      # the values of x, y, and z can be written in one line.
print(x, y, z)
print(x)
print(y)
print(z)
print(x/y)
print(x/z)
print(y/z)
print(x+y+z)
print(x*y*z)
print(x-y-z)
print(x/y/z)
print(x//y//z)
print(x%y%z)

```

```
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]:

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

Out[1]:

'Hello World!'

In [2]:

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

Out[2]:

'Hello World!'

In [3]:

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

Out[3]:

'3 6 9 2 6 8'

In [4]:

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

Out[4]:

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

In [5]:

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

Hello World!

In [6]:

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

Hello World!

Out[6]:

'Hello World!'

## Indexing of a string

In [7]:

```
# printing the first element in a string

message = 'Hello World!'
print(message[0])
```

H

In [8]:

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

r

In [9]:

```
# lenght of a string includign spaces

len(message)
```

Out[9]:

12

In [10]:

```
# Printing the last element in a string
print(message[11])

# Another comment writing type is as follows using triple quotes.

"""
Although the length of the string is 12, since the indexing in Python starts with 0,
the number of the last element is therefore 11.
"""
```

!

Out[10]:

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

## Negative indexing of a string

In [11]:

```
# printing the last element of a string

message[-1]
```

Out[11]:

'!'

In [12]:

```
# printing the first element of a string

message[-12]

"""
Since the negative indexing starts with -1, in this case, the negative index number
of the first element is equal to -12.
"""
```

Out[12]:

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

In [13]:

```
print(len(message))
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]:

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

Out[19]:

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

In [20]:

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

Out[20]:

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

## Escape sequences

In [21]:

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

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

In [22]:

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

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

In [23]:

```
# back slash in a string
print('Hello World! \\ How many people are living on the earth?')

# r will say python that a string will be show as a raw string
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]:

```
message = 'hello python!'
print('Before uppercase: ', message )

# convert uppercase the elements in a string
message_upper = message.upper()
print('After uppercase: ', message_upper)

# convert lowercase the elements in a string
message_lower = message.lower()
print('Again lowercase: ', message_lower)

# convert first letter of string to uppercase
message_title = message.title()
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]:

```
# replace() method in a string
message = 'Hello Python!'
message_hi = message.replace('Hello', 'Hi')
message_python = message.replace('Python', 'World')
print(message_hi)
print(message_python)
```

```
Hi Python!
Hello World!
```

In [26]:

```
# find() method application in a string
message = 'Hello World!'
print(message.find('Wo'))

# the output is the index number of the first element of the substring
```

```
6
```

In [27]:

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

Out[27]:

```
6
```

In [28]:

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

Out[28]:

-1

In [30]:

```
text = 'Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. Had

# find the first index of the substring 'Nancy'
text.find('Nancy')
```

Out[30]:

122

In [31]:

```
# replace the substring 'Nancy' with 'Nancy Lier Cosgrove Mullis'
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, someda y, might get very lucky, and make his own heaven out of one of the people around him. She will be his m orning 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 pr ecede the irises. Their faith in one another will be deeper than time and their eternal spirit will be seaml ess once again.'

In [32]:

```
# convet the text to lower case
text.lower()
```

Out[32]:

'jean-paul sartre somewhere observed that we each of us make our own hell out of the people around u s. 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 f aith in one another will be deeper than time and their eternal spirit will be seamless once again.'

In [33]:

```
# convert the first letter of the text to capital letter
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 luck y, and make his own heaven out of one of the people around him. she will be his morning and his evenin g star, shining with the brightest and the softest light in his heaven. she will be the end of his wandering s, and their love will arouse the daffodils in the spring to follow the crocuses and precede the irises. thei r faith in one another will be deeper than time and their eternal spirit will be seamless once again.'

In [34]:

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

Out[34]:

'jean-paul sartre somewhere observed that we each of us make our own hell out of the people around u s. 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 f aith in one another will be deeper than time and their eternal spirit will be seamless once again.'

In [35]:

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

Out[35]:

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

In [36]:

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

Out[36]:

7

In [37]:

```
# format() method
"""
The format() method formats the specified value(s) and insert them inside the string's placeholder.
The placeholder is defined using curly brackets: {}.
"""

txt = "Hello {word}"
print(txt.format(word = 'World!'))

message1 = 'Hi, My name is {} and I am {} years old.'
print(message1.format('Bob', 36))

message2 = 'Hi, My name is {name} and I am {number} years old.'
print(message2.format(name ='Bob', number = 36))

message3 = 'Hi, My name is {0} and I am {1} years old.'
print(message3.format('Bob', 36))
```

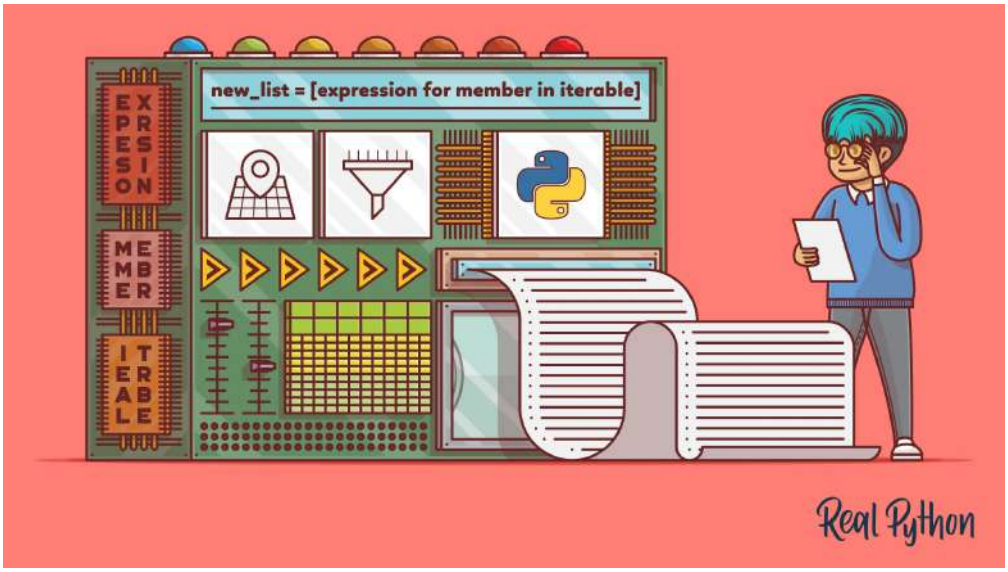
```
Hello World!
Hi, My name is Bob and I am 36 years old.
Hi, My name is Bob and I am 36 years old.
Hi, My name is Bob and I am 36 years old.
```

# 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]:

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

Out[1]:

['python', 25, 2022]

In [7]:

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

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

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

Out[10]:

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

In [11]:

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

Out[11]:

```
5
```

### Slicing

In [20]:

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

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

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

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

```
nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
print('Before changing:', nlis)
nlis[0] = 'hello python!'
print('After changing:', nlis)
nlis[1] = 1.618
print('After changing:', nlis)
nlis[2] = [3.14, 2022]
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, 2
022)]
```

### Deleting the element from the list using del() function

In [34]:

```
print('Before changing:', nlis)
del(nlis[0])
print('After changing:', nlis)
del(nlis[-1])
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]:

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

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

Out[36]:

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

### Use of split() function with a delimiter

In [57]:

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

Out[57]:

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

### Basic operations

In [90]:

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

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

```
# The element in the copied list also changes when the element in the original list was changed.
# See the following example

nlis = ['python', 3.14, 2022, [1, 1, 2, 3, 5, 8, 13, 21, 34], ('hello', 'python', 3,14, 2022)]
print(nlis)
copy_list = nlis
print(copy_list)
print('copy_list[0]:', copy_list[0])
nlis[0] = 'hello python!'
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]:

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

Out[72]:

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

In [74]:

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

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

```
text = input('Enter a string:')
print('The text is', text)
print(type(text))
```

```
The text is Hello, Python!
<class 'str'>
```

In [12]:

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

```
The number is 15
<class 'str'>
```

In [15]:

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

```
The number is 15
<class 'int'>
```

In [16]:

```
number = float(input('Enter an integer:'))
print('The number is', number)
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]:

```
expression = '8+7'
total = eval(expression)
print('Sum of the expression is', total)
print(type(expression))
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 secreen with good look and attractive.

In [22]:

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

```
Sum of 3.14 and 1.618 is 4.758.
```

In [25]:

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

```
a = 3.14
b = 1.618
print('a>b is:', a>b)
print('a<b is:', a<b)
print('a<=b is:', a<=b)
print('a>=b is:', a>=b)
print('a==b is:', a==b)
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]:

```
a = 3.14
b = 1.618
c = 12
d = 3.14
print(a>b and c>a)
print(b>c and d>a)
print(b<c or d>a)
print( not a==b)
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]:

```
x = 3.14
x+=5
print(x)
```

8.14

In [43]:

```
x = 3.14
x-=5
print(x)
```

-1.8599999999999999

In [44]:

```
x = 3.14
x*=5
print(x)
```

15.700000000000001

In [45]:

```
x = 3.14
x/=5
print(x)
```

0.628

In [46]:

```
x = 3.14
x%=5
print(x)
```

3.14

In [47]:

```
x = 3.14
x//=5
print(x)
```

0.0

In [48]:

```
x = 3.14
x**=5
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 momory location and return *True* or *False*.

In [74]:

```
a = 3.14
b = 1.618
print(a is b)
print(a is not b)
msg1= 'Hello, Python!'
msg2 = 'Hello, World!'
print(msg1 is msg2)
print(msg1 is not msg2)
lis1 = [3.14, 1.618]
lis2 = [3.14, 1.618]
print(lis1 is lis2)          # You should see a list copy behavior
print(lis1 is not lis2)
```

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

### Membership operators

- These operators inclusing **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]:

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

```
False
False
True
True
```

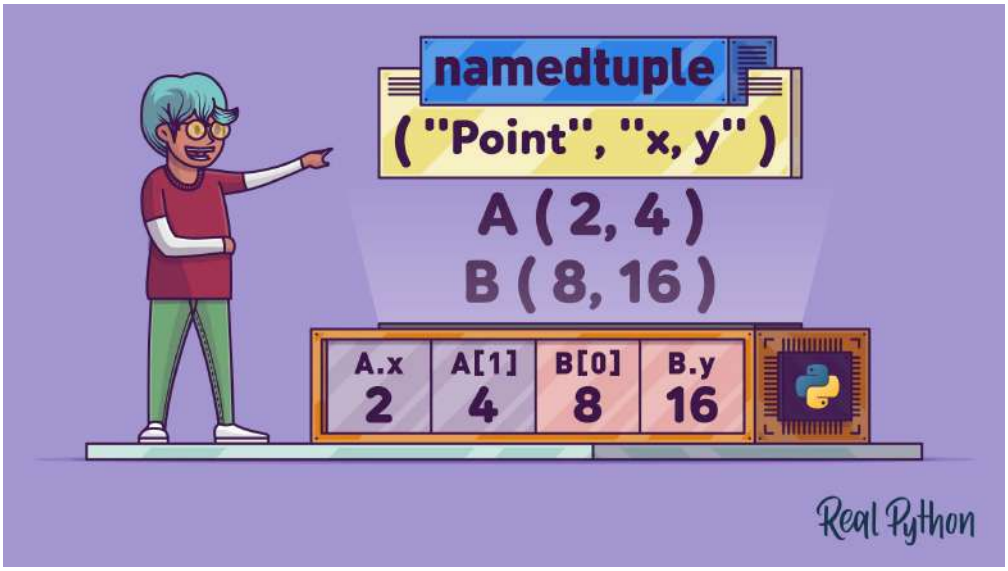
# Python Tutorial

**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]:

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

```
print(type(tuple_1))
print(len(tuple_1))
```

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

### Indexing

In [12]:

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

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

In [11]:

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

```
tuple_2 = tuple_1 + ('Hello World!', 2022)
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]:

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

Out[48]:

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

### Membership

In [49]:

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

```

```
True
False
False
True
```

### Iteration

In [50]:

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

```
1
2
3
4
```

### cmp() function

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

In [55]:

```
def cmp(t1, t2):
    return bool(t1 > t2) - bool(t1 < t2)
def cmp(t31, t4):
    return bool(t3 > t4) - bool(t3 < t4)
def cmp(t5, t6):
    return bool(t5 > t6) - bool(t5 < t6)
t1 = (1,3,5)          # Here t1 is lower than t2, since the output is -1
t2 = (2,4,6)

t3 = (5,)             # Here t3 is higher than t4 since the output is 1
t4 = (4,)

t5 = (3.14,)          # Here t5 is equal to t6 since the output is 0
t6 = (3.14,)

print(cmp(t1, t2))
print(cmp(t3, t4))
print(cmp(t5, t6))
```

```
-1
1
0
```

### min() function

In [56]:

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

Out[56]:

```
1
```

### max() function

In [58]:

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

Out[58]:

4

### tup(seq) function

It converts a specific sequence to a tuple

In [60]:

```
seq = 'ATGCGTATTGCCAT'
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]:

```
# Obtaining a new tuple from the index 2 to index 6

tuple_1 = ('Hello', 'Python', 3.14, 1.618, True, False, 32, [1,2,3], {1,2,3}, {'A': 3, 'B': 8}, (0, 1))
tuple_1[2:7]
```

Out[14]:

(3.14, 1.618, True, False, 32)

In [18]:

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

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

Out[19]:

11

### Sorting tuple

In [22]:

```
# Tuples can be sorted and save as a new tuple.

tuple_3 = (0,9,7,4,6,2,9,8,3,1)
sorted_tuple_3 = sorted(tuple_3)
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]:

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

Out[25]:

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

In [26]:

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

```
# Using second index to access other tuples in the nested tuple
print('Item 1, 0 of the nested tuple is', nested_tuple[1][0])
print('Item 1, 1 of the nested tuple is', nested_tuple[1][1])
print('Item 2, 0 of the nested tuple is', nested_tuple[2][0])
print('Item 2, 1 of the nested tuple is', nested_tuple[2][1])
print('Item 3, 0 of the nested tuple is', nested_tuple[3][0])
print('Item 3, 1 of the nested tuple is', nested_tuple[3][1])
print('Item 3, 2 of the nested tuple is', nested_tuple[3][2])

# Accesing to the items in the second nested tuples using a third index
print('Item 3, 2, 0 of the nested tuple is', nested_tuple[3][2][0])
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]:

```
# Take a tuple
tuple_4 = (1,3,5,7,8)
tuple_4[0] = 9
print(tuple_4)

# 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]:

```
tuple_4 = (1,3,5,7,8)
print('Before deleting:', tuple_4)
del tuple_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]:

```
tuple_5 = (1,1,3,3,5,5,5,5,6,6,7,8,9)
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]:

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

```
tuple_6 = (0)
print(tuple_6)
print(type(tuple_6))

# Here, you see that the output is an integer
```

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

In [47]:

```
tuple_7 = (0,)
print(tuple_7)
print(type(tuple_7))

# You see that the output is a tuple
```

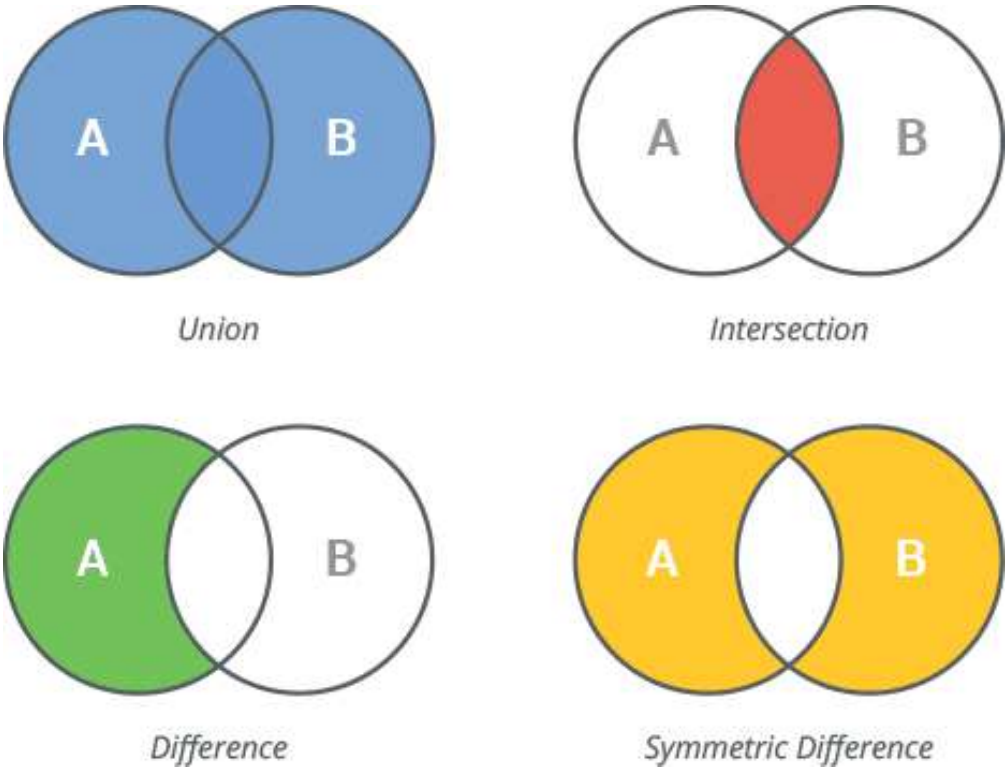
```
(0,)
<class 'tuple'>
```

# 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 **{}**.




In [47]:

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

<class 'dict'>

In [46]:

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

<class 'set'>

In [2]:

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

Out[2]:

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

## Converting list to set

In [4]:

```
# A list can convert to a set
# Take a list
nlis = ['Hello Python!', 3.14, 1.618, 'Hello World!', 3.14, 1.618, True, False, 2022]

# Convert the list to a set
set2 = set(nlis)
set2
```

Out[4]:

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

## Set operations

In [5]:

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

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

Out[6]:

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

In [7]:

```
# Addition of the same element
set3.add('Hi, Python!')
set3

# 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]:

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

```
set3.remove('Hello Python!')
set3

```

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]:

```
set3.discard(3.14)
set3
```

Out[50]:

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

In [17]:

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

Out[17]:

True

## Logic operations in Sets

In [18]:

```
# Take two sets
set4 = set(['Hello Python!', 3.14, 1.618, 'Hello World!'])
set5 = set([3.14, 1.618, True, False, 2022])

# Printing two sets
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]:

```
intersection = set4 & set5
intersection
```

Out[19]:

{1.618, 3.14}

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

In [21]:

```
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]:

```
print(set4.difference(set5))
print(set5.difference(set4))

# The same process can make using subtraction operator as follows:
print(set4-set5)
print(set5-set4)
```

{'Hello Python!', 'Hello World!'}
{False, True, 2022}
{'Hello Python!', 'Hello World!'}
{False, True, 2022}

### Set comparison

In [62]:

```
print(set4>set5)
print(set5>set4)
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]:

```
A = [1,1,2,2,3,3,4,4,5,5]      # Take a list
B = {1,1,2,2,3,3,4,4,5,5}      # Take a set

print('The minimum number of A is', min(A))
print('The minimum number of B is', min(B))
print('The maximum number of A is', max(A))
print('The maximum number of B is', max(B))
print('The sum of A is', sum(A))
print('The sum of B is', sum(B))

# 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]:

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

```
set7 = {1,2,3,4}
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]:

```
set8 = {1,3,5,7,9}
print(set8)
set9 = set8
print(set9)
set8.add(11)
print(set8)
print(set9)

"""
As you see that although the number 8 is added into the set 'set8', the added number
is also added into the set 'set9'
"""
```

```
{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]:

```
set8 = {1,3,5,7,9}
print(set8)
set9 = set8.copy()
print(set9)
set8.add(11)
print(set8)
print(set9)

"""
When this function is used, the original set stays unmodified.
A new copy stored in another set of memory locations is created.
The change made in one copy won't reflect in another.
"""
```

```
{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"
```

### celar() function

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

In [57]:

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

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

### pop() function

It removes and returns an arbitrary set element.

In [60]:

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

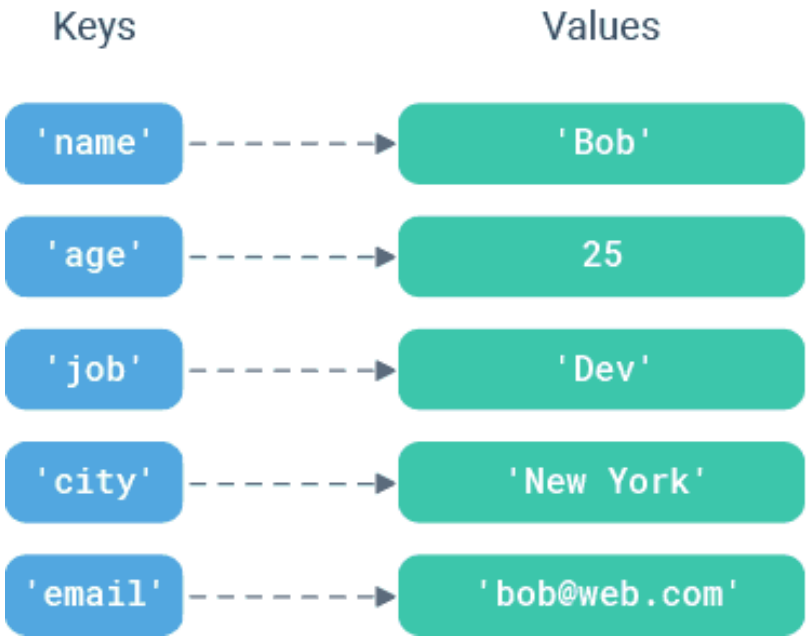
```
{0, 1, 2, 3, 34, 5, 8, 13, 21}
{1, 2, 3, 34, 5, 8, 13, 21}
```

# 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]:

```
# Take a sample dictionary

sample_dict = {'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'}
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]:

```
# Accessing to the value using the key
print(sample_dict['key_1'])
print(sample_dict['key_2'])
print(sample_dict['key_3'])
print(sample_dict['key_4'])
print(sample_dict['key_5'])
print(sample_dict['key_6'])
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]:

```
# Take a sample dictionary
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'}
product

```

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]:

```
# Retrieving the value by keys
print(product['Aspergillus niger'])
print(product['Saccharomyces cerevisiae'])
print(product['Scheffersomyces stipitis'])
```

```
inulinase
ethanol
ethanol
```

### keys() function to get the keys in the dictionary

In [28]:

```
# What are the keys in the dictionary?
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]:

```
# What are the values in the dictionary?
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]:

```
product['Yarrovia lipolytica'] = 'microbial oil'
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',
 'Yarrovia lipolytica': 'microbial oil'}
```

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

In [32]:

```
del(product['Aspergillus niger'])
del(product['Aspergillus sojae_1'])
product
```

Out[32]:

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

In [1]:

```
del product
print(product)

# 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]:

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

```
True
False
```

### dict() function

This function is used to create a dictionary

In [19]:

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

Out[19]:

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

In [21]:

```
# Numerical index is not used to take the dictionary values. It gives a KeyError
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]:

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

Out[34]:

{}

### copy() function

It returns a shallow copy of the main dictionary

In [35]:

```
sample_original = dict(family = 'music', type='pop', year='2022' , name='happy new year')
sample_copy = sample_original.copy()
print(sample_original)
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]:

```
# This method can be made usign '=' sign
sample_copy = sample_original
print(sample_copy)
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]:

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

```

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

### popitem() function

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

In [39]:

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

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

```
keys = {'A', 'T', 'C', 'G'}
sequence = dict.fromkeys(keys)
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]:

```
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'}

sample_original = dict(family = 'music', type='pop', year='2022' , name='happy new year')

product.update(sample_original)
print(product)
```

```
{'Aspergillus niger': 'inulinase', 'Saccharomyces cerevisiae': 'ethanol', 'Scheffersomyces stipitis': 'ethanol',
'Aspergillus sojae_1': 'mannanase', 'Streptococcus zooepidemicus': 'hyaluronic acid', 'Lactobacillus case
i': 'lactic acid', 'Aspergillus sojae_2': 'polygalacturonase', 'family': 'music', 'type': 'pop', 'year': '2022', 'na
me': '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]:

```
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'}

product.items()
```

Out[46]:

```
dict_items([('Aspergillus niger', 'inulinase'), ('Saccharomyces cerevisiae', 'ethanol'), ('Scheffersomyces sti
pitis', '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]:

```
# 'for' loop print all the keys in the dictionary

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'}

for k in product:
    print(k)
```

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

In [15]:

```
# 'for' loop to print the values of the dictionary by using values() and other method

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'}
for x in product.values():
    print(x)

print()
# 'for' loop to print the values of the dictionary by using values() and other method
for x in product:
    print(product[x])
```

```
inulinase
ethanol
ethanol
mannanase
hyaluronic acid
lactic acid
polygalacturonase

inulinase
ethanol
ethanol
mannanase
hyaluronic acid
lactic acid
polygalacturonase
```

In [16]:

```
# 'for' loop to print the items of the dictionary by using items() method
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'}

for x in product.items():
  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]:

```
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'}

for x, y in product.items():
  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]:

```
# Take a variable
golden_ratio = 1.618

# Condition less than
print(golden_ratio<2)     # The golden ratio is lower than 2, thus the output is True
print(golden_ratio<1)     # The golden ratio is greater than 1, thus the output is False
```

```
True
False
```

In [4]:

```
# Take a variable
golden_ratio = 1.618

# Condition less than or equal to
print(golden_ratio<=2)     # The golden ratio is lower than 2, thus the condition is True.
print(golden_ratio<=1)     # The golden ratio is greater than 1, thus the condition is False.
print(golden_ratio<=1.618)  # The golden ratio is equal to 1.618, thus the condition is True.
```

```
True
False
True
```

In [5]:

```
# Take a variable
golden_ratio = 1.618

# Condition greater than
print(golden_ratio>2)     # The golden ratio is lower than 2, thus the condition is False.
print(golden_ratio>1)     # The golden ratio is greater than 1, thus the condition is True.
```

```
False
True
```

In [7]:

```
# Take a variable
golden_ratio = 1.618

# Condition greater than or equal to
print(golden_ratio>=2)  # The golden ratio is not greater than 2, thus the condition is False.
print(golden_ratio>=1)  # The golden ratio is greater than 1, thus the condition is True.
print(golden_ratio>=1.618)  # The golden ratio is equal to 1.618, thus the condition is True.
```

```
False
True
True
```

In [8]:

```
# Take a variable
golden_ratio = 1.618

# Condition equal to
print(golden_ratio==2)  # The golden ratio is not equal to 1.618, thus the condition is False.
print(golden_ratio==1.618)  # The golden ratio is equal to 1.618, thus the condition is True.
```

```
False
True
```

In [11]:

```
# Take a variable
golden_ratio = 1.618

# Condition not equal to
print(golden_ratio!=2)  # The golden ratio is not equal to 1.618, thus the condition is True.
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/) value of letters.

In [17]:

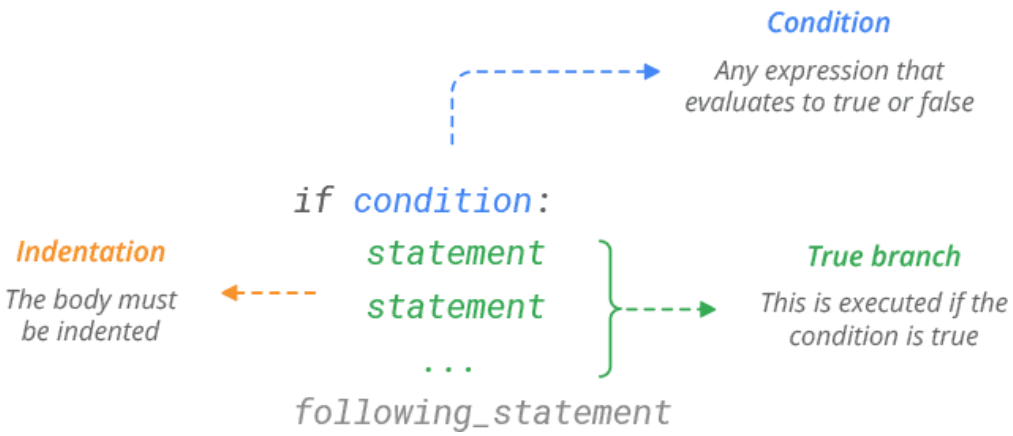
```
# Compare strings
print('Hello' == 'Python')
print('Hello' != 'Python')
print('Hello' <= 'Python')
print('Hello' >= 'Python')
print('Hello' < 'Python')
print('Hello' > 'Python')
print('B'>'A')  # According to ASCII table, the values of A and B are equal 65 and 66, respectively.
print('a'>'b')  # According to ASCII table, the values of a and b are equal 97 and 98, respectively.
print('CD'>'DC') # According to ASCII table, the value of C (67) is lower than that of D (68)

# 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]:

```
pi = 3.14
golden_ratio = 1.618

# This statement can be True or False.
if pi > golden_ratio:

    # If the conditions is True, the following statement will be printed.
    print(f'The number pi {pi} is greater than the golden ratio {golden_ratio}.')

# The following statement will be printed in each situtation.
print('Done!')
```

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

In [2]:

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

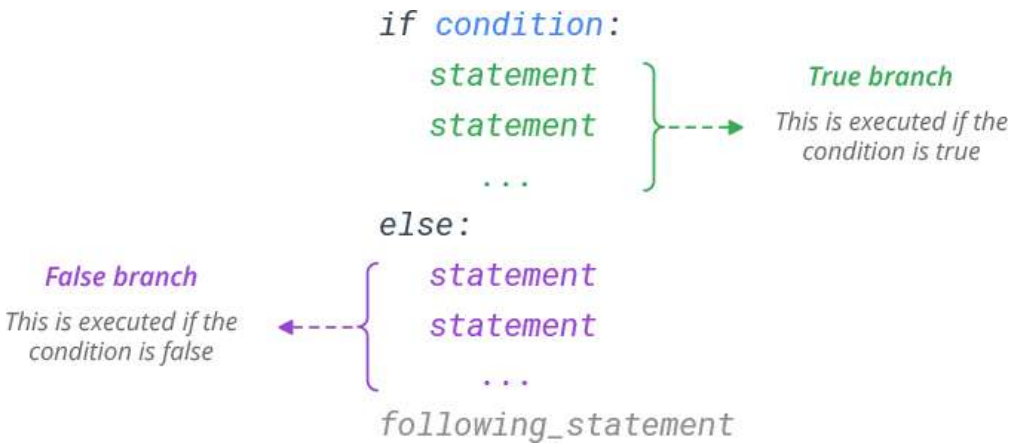
Hello, python!

In [5]:

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

This is true.

### else statement

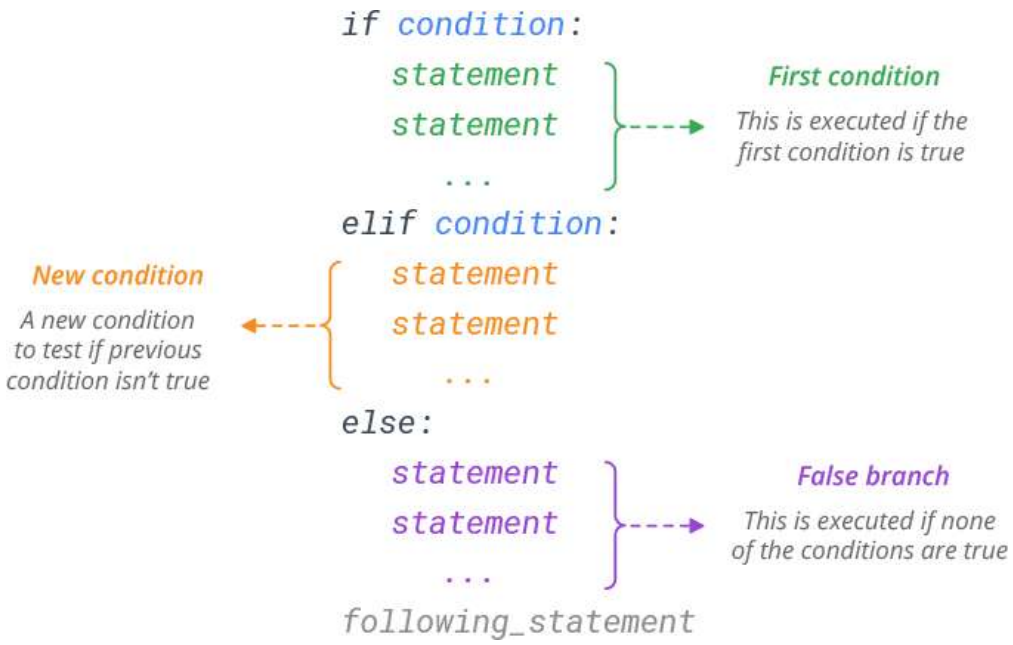


In [8]:

```
pi = 3.14
golden_ratio = 1.618

if pi < golden_ratio:
    print(f'The number pi {pi} is greater than the golden ratio {golden_ratio}.')
else:
    print(f'The golden ratio {golden_ratio} is lower than the number pi {pi}.')
print('Done!')
```

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

### elif statement




In [23]:

```
age = 5

if age > 6:
    print('You can go to primary school.' )
elif age == 5:
    print('You should go to kindergarten.')
else:
    print('You are a baby' )

print('Done!')
```

You should go to kindergarten.
Done!

In [25]:

```
album_year = 2000
album_year = 1990

if album_year >= 1995:
    print('Album year is higher than 1995.')

print('Done!')
```

Done!

In [26]:

```
album_year = 2000
# album_year = 1990

if album_year >= 1995:
    print('Album year is higher than 1995.')
else:
    print('Album year is lower than 1995.')

print('Done!')
```

Album year is higher than 1995.
Done!

In [43]:

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

The movie could win Oscar.

In [13]:

```
movie_rating = float(input('Enter a rating number:'))

print(f'The entered movie rating is: {movie_rating}')

if movie_rating > 8.5:
    print('The movie is awesome with {} rating and you should watch it.'.format(movie_rating))
else:
    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]:

```
note = float(input('Enter a note:'))

print(f'The entered note value is: {note}')

if note >= 90 and note <= 100:
    print('The letter grade is AA.')
elif note >= 85 and note <= 89:
    print('The letter grade is BA.')
elif note >= 80 and note <= 84:
    print('The letter grade is BB.')
elif note >= 75 and note <= 79:
    print('The letter grade is CB.')
elif note >= 70 and note <= 74:
    print('The letter grade is CC.')
elif note >= 65 and note <= 69:
    print('The letter grade is DC.')
elif note >= 60 and note <= 64:
    print('The letter grade is DD.')
elif note >=55 and note <= 59:
    print('The letter grade is ED.')
elif note >=50 and note <= 54:
    print('The letter grade is EE.')
elif note >=45 and note <=49:
    print('The letter grade is FE.')
else:
    print('The letter grade is FF.')
```

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

In [17]:

```
number = int(input('Enter a number:'))

print(f'The entered number is: {number}')

if number %2 == 0:
    print(f'The entered number {number} is even')
else:
    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]:

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

```
You were born between 1990 and 1994
Done!
```

In [23]:

```
x = int(input('Enter a number:'))
y = int(input('Enter a number: '))
z = int(input('Enter a number:'))

print(f'The entered numbers for x, y, and z are {x}, {y}, and {z}, respectively.')

if x>y and x>z:
    print(f'The number x with {x} is the greatest number.')
elif y>x and y>z:
    print(f'The number y with {y} is the greatest number.')
else:
    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]:

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

```
You were not born in 1980s.
Done!
```

### not

In [29]:

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

The year of birth is not 1991.

In [15]:

```
birth_year = int(input('Enter a year of birth: '))

print(f'The entered year of birth is: {birth_year}')

if birth_year < 1985 or birth_year == 1991 or birth_year == 1995:
    print(f'You were born in {birth_year}')
else:
    # For instance, if your year of birth is 1993
    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]:

```
birth_year = int(input('Enter a year of birth: '))

print(f'The entered year of birth is: {birth_year}')

if birth_year < 1985 or birth_year == 1991 or birth_year == 1995:
    # For instance, if your year of birth is 1995
    print(f'You were born in {birth_year}')
else:
    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.

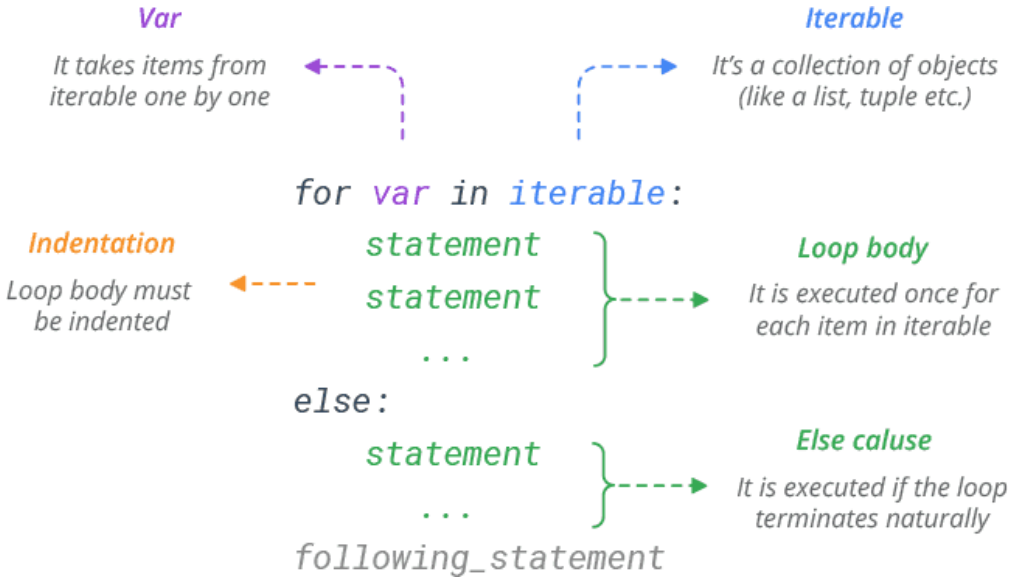
In [3]:

```
# Take a range() function
print(range(5))
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]:

```
# Take an example
# Diectly accessing to the elements in the list

years = [2005, 2006, 2007, 2008, 2009, 2010]

for i in years:
    print(i)
```

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

In [10]:

```
# Again, directly accessing to the elements in the list
years = [2005, 2006, 2007, 2008, 2009, 2010]

for year in years:
    print(year)
```

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

In [6]:

```
# Take an example
years = [2005, 2006, 2007, 2008, 2009, 2010]

for i in range(len(years)):
    print(years[i])
```

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

In [8]:

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

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

In [16]:

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

```
2
5
8
11
```

In [12]:

```
# Changing the elements in the list
languages = ['Java', 'JavaScript', 'C', 'C++', 'PHP']

for i in range(len(languages)):
    print('Before language', i, 'is', languages[i])
    languages[i] = 'Python'
    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]:

```
# Enumaeration of the elements in the list
languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']

for index, language in enumerate(languages):
    print(index, language)
```

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

In [30]:

```
# Take the numbers between -3 and 6 using for loop
# Use range() function

for i in range(-3, 7):
    print(i)
```

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

In [31]:

```
# Take a list and print the elements using for loop
languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']

for i in range(len(languages)):
    print(i, languages[i])
```

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

In [120]:

```
number1 = int(input('Enter a number:'))
number2 = int(input('Enter a number:'))
print(f'The entered numbers are {number1} and {number2}.')
for i in range(0, 11):
    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]:

```
# Take a list
nlis = [0.577, 2.718, 3.14, 1.618, 1729, 6, 37]

# Write a for loop for addition
count = 0
for i in nlis:
    count+=i
print('The total value of the numbers in the list is', count)

# Calculate the average using len() function
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 [19]:

```
for i in range(1,6):
    print(i, end=", ")
else:
    print('These are numbers from 1 to 5.')
```

1, 2, 3, 4, 5, These are numbers from 1 to 5.

## nested for loop

In [112]:

```
num = int(input('Enter a number:'))

print(f'The entered the number is {num}.')
i, j = 0, 0
for i in range(0, num):
    print()
    for j in range(0, i+1):
        print('+', end='')
```

The entered the number is 10.

```
+
++
+++
++++
+++++
++++++
+++++++
++++++++
+++++++++
++++++++++
```

## continue in for loop

In [116]:

```
# Take a list
nlis = [1,2,4,5,6,7,8,9,10,11,12,13,14]
for i in nlis:
    if i == 5:
        continue
    print(i)

"""
You see that the output includes the numbers without 5.
The continue function jumps when it meets with the reference.
"""
```

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

Out[116]:

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

## break in for loop

In [118]:

```
# Take a list
nlis = [1,2,4,5,6,7,8,9,10,11,12,13,14]
for i in nlis:
    if i == 5:
        break
    print(i)

"""
You see that the output includes the numbers before 5.
The break function terminate the loop when it meets with the reference.
"""
```

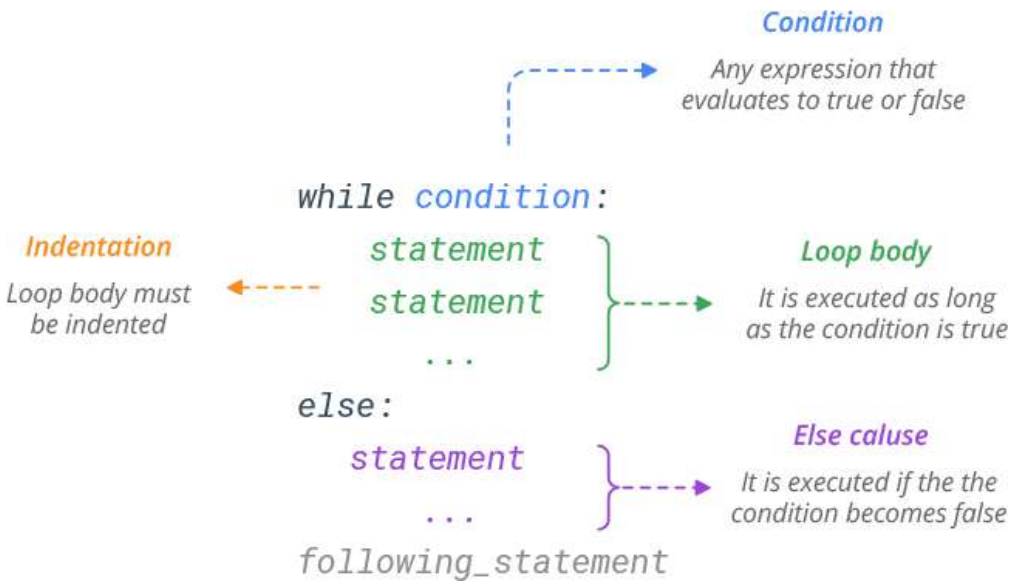
```
1
2
4
```

Out[118]:

'\nYou see that the output includes the numbers before 5. \nThe break function terminate the loop whe
n 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]:

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

```
22
23
24
25
26
```

In [22]:

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

```
22
21
20
19
18
17
```

In [25]:

```
# Take an example
years = [2005, 2006, 2007, 2008, 2009, 2010]

index = 0

year = years[0]

while year !=2008:
    print(year)
    index+=1
    year = years[index]
print('It gives us only', index, 'repetititons to get out of loop')

```

2005
2006
2007
It gives us only 3 repetititons to get out of loop

In [37]:

```
# Print the movie ratings gretater than 6.
movie_rating = [8.0, 7.5, 5.4, 9.1, 6.3, 6.5, 2.1, 4.8, 3.3]

index = 0
rating = movie_rating[0]

while rating>=6:
    print(rating)
    index += 1
    rating = movie_rating[index]
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]:

```
# Print the movie ratings gretater than 6.
movie_rating = [8.0, 7.5, 5.4, 9.1, 6.3, 6.5, 2.1, 4.8, 3.3]

index = 0
for i in range(len(movie_rating)):
    if movie_rating[i] >= 6:
        index += 1
        print(index, movie_rating[i])
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]:

```
# Adding the element in a list to a new list
fruits = ['banana', 'apple', 'banana', 'orange', 'kiwi', 'banana', 'Cherry', 'Grapes']

new_fruits = []

index = 0
while fruits[index] == 'banana':
    new_fruits.append(fruits[index])
    index += 1
print(new_fruits)
```

['banana']

In [119]:

```
number1 = int(input('Enter a number:'))
number2 = int(input('Enter a number:'))
print(f'The entered numbers are {number1} and {number2}.')

i = 0
while i<=10:
    print(('%d x %d = %d' %(number1, i, number1*i)), ',', ('%d x %d = %d' %(number2, i, number2*i )))
    i+=1

```

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]:

```
index = 0
while index <=5:
    print(index, end=' ')
    index += 1
else:
    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]:

```
i = 0

while i<=5:
    print(i)
    i+=1
    if i == 3:
        continue
```

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

## break in while loop

In [121]:

```
i = 0

while i<=5:
    print(i)
    i+=1
    if i == 3:
        break
```

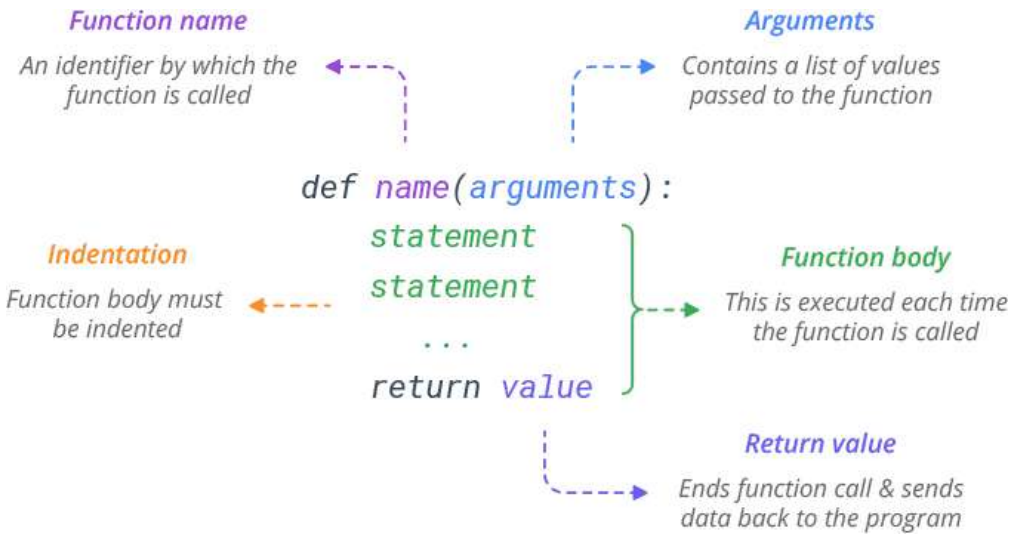
```
0
1
2
```

# 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]:

```
# Take a function sample
# Mathematical operations in a function

def process(x):
    y1 = x-8
    y2 = x+8
    y3 = x*8
    y4 = x/8
    y5 = x%8
    y6 = x//8
    print(f'If you make the above operations with {x}, the results will be {y1}, {y2}, {y3}, {y4}, {y5}, {y6}.')
    return y1, y2, y3, y4, y5, y6

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]:

```
help(process)
```

Help on function process in module __main__:

process(x)

## Call the function again with the number 3.14

In [11]:

```
process(3.14)
```

If you make the above operations with 3.14, the results will be -4.859999999999999, 11.14, 25.12, 0.39
25, 3.14, 0.0.

Out[11]:

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

## Functions with multiple parameters

In [2]:

```
# Define a function with multiple elements
def mult(x, y):
    z = 2*x + 5*y + 45
    return z

output = mult(3.14, 1.618)     # You can yield the output by assigning to a variable
print(output)
print(mult(3.14, 1.618))      # You can obtain the result directly
mult(3.14, 1.618)             # This is also another version
```

```
59.370000000000005
59.370000000000005
```

Out[2]:

```
59.370000000000005
```

In [20]:

```
# Call again the defined function with different arguments
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]:

```
# Define a function
def function(x):

    # Take a local variable
    y = 3.14
    z = 3*x + 1.618*y
    print(f'If you make the above operations with {x}, the results will be {z}.')
    return z

with_golden_ratio = function(1.618)
print(with_golden_ratio)
```

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

In [8]:

```
# It starts the gloabal variable
a = 3.14

# call function and return function
y = function(a)
print(y)
```

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

In [9]:

```
# Enter a number directly as a parameter
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]:

```
# Define a function with and without return statement
def msg1():
    print('Hello, Python!')

def msg2():
    print('Hello, World!')
    return None

msg1()
msg2()
```

Hello, Python!
Hello, World!

In [15]:

```
# Printing the function after a call indicates a None is the default return statement.
# See the following prontings what functions returns are.

print(msg1())
print(msg2())
```

Hello, Python!
None
Hello, World!
None

### Concatetantion of two strings

In [18]:

```
# Define a function
def  strings(x, y):
    return x + y

# Testing the function 'strings(x, y)'
strings('Hello', ' ' 'Python')
```

Out[18]:

'Hello Python'

### Simplicity of functions

In [26]:

```
# The following codes are not used again.
x = 2.718
y = 0.577
equation = x*y + x+y - 37
if equation>0:
    equation = 6
else: equation = 37

equation
```

Out[26]:

37

In [27]:

```
# The following codes are not used again.
x = 0
y = 0
equation = x*y + x+y - 37
if equation<0:
    equation = 0
else: equation = 37

equation
```

Out[27]:

0

In [28]:

```
# The following codes can be write as a function.
def function(x, y):
    equation = x*y + x+y - 37
    if equation>0:
        equation = 6
    else: equation = 37
    return equation

x = 2.718
y = 0.577
function(x, y)
```

Out[28]:

37

In [29]:

```
# The following codes can be write as a function.
def function(x, y):
    equation = x*y + x+y - 37
    if equation<0:
        equation = 6
    else: equation = 37
    return equation

x = 0
y = 0
function(x, y)
```

Out[29]:

6

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

In [31]:

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

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

In [32]:

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

Out[32]:

1808.053

In [33]:

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

Out[33]:

8

### Using conditions and loops in functions

In [44]:

```
# Define a function including conditions if/else

def fermentation(microorganism, substrate, product, activity):
    print(microorganism, substrate, product, activity)
    if activity < 1000:
        return f'The fermentation process was unsuccessful with the {product} activity of {activity} U/mL from {substrate}
    else:
        return f'The fermentation process was successful with the {product} activity of {activity} U/mL from {substrate} us

result1 = fermentation('Aspergillus niger', 'molasses', 'inulinase', 1800)
print(result1)
print()
result2 = fermentation('Aspergillus niger', 'molasses', 'inulinase', 785)
print(result2)

```

Aspergillus niger molasses inulinase 1800
The fermentation process was successful with the inulinase activity of 1800 U/mL from molasses using A spergillus 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]:

```
# Define a function using the loop 'for'

def fermentation(content):
    for parameters in content:
        print(parameters)

content = ['Stirred-tank bioreactor' ,'30°C temperature', '200 rpm agitation speed', '1 vvm aeration', '1% (v/v) inoculur
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

### Adjustiing default values of independent variables in functions

In [53]:

```
# Define a function adjusting the default value of the variable

def rating_value(rating = 5.5):
    if rating < 8:
        return f'You should not watch this film with the rating value of {rating}'
    else:
        return f'You should watch this film with the rating value of {rating}'

print(rating_value())
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]:

```
# Define a function for a global variable
language = 'Python'

def lang(language):
    global_var = language
    print(f'{language} is a program language.')

lang(language)
lang(global_var)

"""
The output gives a NameError, since all variables in the function are local variables,
so variable assignment is not persistent outside the function.
"""
```

```
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]:

```
# Define a function for a global variable
language = 'JavaScript'

def lang(language):
    global global_var
    global_var = 'Python'
    print(f'{language} is a programing language.')

lang(language)
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]:

```
process = 'Continuous fermentation'

def fermentation(process_name):
    if process_name == process:
        return '0.5 g/L/h.'
    else:
        return '0.25 g/L/h.'

print('The productiovity in continuous fermentation is', fermentation('Continuous fermentation'))
print('The productiovity in batch fermentation is', fermentation('Batch fermentation'))
print('Continuous fermentation has many advantages over batch fermentation.')
print(f'My favourite process is {process}.')
```

The productiovity in continuous fermentation is 0.5 g/L/h.
The productiovity 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]:

```
# If the variable 'process' is deleted, it returns a NameError as follows
del process

# Since the variable 'process' is deleted, the following function is an example of local variable
def fermentation(process_name):
    process = 'Continuous fermentation'
    if process_name == process:
        return '0.5 g/L/h.'
    else:
        return '0.25 g/L/h.'

print('The productiovity in continuous fermentation is', fermentation('Continuous fermentation'))
print('The productiovity in batch fermentation is', fermentation('Batch fermentation'))
print('Continuous fermentation has many advantages over batch fermentation.')
print(f'My favourite process is {process}.')
```

```
The productiovity in continuous fermentation is 0.5 g/L/h.
The productiovity 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 productiovity 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]:

```
# When the global variable and local variable have the same name:

process = 'Continuous fermentation'

def fermentation(process_name):
    process = 'Batch fermentation'
    if process_name == process:
        return '0.5 g/L/h.'
    else:
        return '0.25 g/L/h.'

print('The productiovity in continuous fermentation is', fermentation('Continuous fermentation'))
print('The productiovity in batch fermentation is', fermentation('Batch fermentation'))
print(f'My favourite process is {process}.')
```

```
The productiovity in continuous fermentation is 0.25 g/L/h.
The productiovity 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 unkknown for a function, then the arguments can be packet into a tuple or a dictionary

In [84]:

```
# Define a function regarding a tuple example
def function(*args):
    print('Number of elements is', len(args))
    for element in args:
        print(element)

function('Aspergillus niger', 'inulinase', 'batch', '1800 U/mL activity')
print()
function('Saccharomyces cerevisia', 'ethanol', 'continuous', '45% yield', 'carob')

```

```
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]:

```
# Another example regarding 'args'
def total(*args):
    total = 0
    for i in args:
        total += i
    return total

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]:

```
# Define a function regarding a dictionary example
def function(**args):
    for key in args:
        print(key, ':', args[key])

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]:

```
# Define a function regarding the addition of elements into a list
def addition(nlist):
    nlist.append(3.14)
    nlist.append(1.618)
    nlist.append(1729)
    nlist.append(6)
    nlist.append(37)

my_list= [0.577, 2.718]
addition(my_list)
print(my_list)
print(sum(my_list))
print(min(my_list))
print(max(my_list))
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]:

```
# Define a function
def addition(x, y):
    """The following function returns the sum of two parameters."""
    z = x+y
    return z

print(addition.__doc__)
print(addition(3.14, 2.718))
```

```
The following function returns the sum of two parameters.
5.8580000000000005
```

### Recursive functions

In [103]:

```
# Calculating the factorial of a certain number.

def factorial(number):
    if number == 0:
        return 1
    else:
        return number*factorial(number-1)

print('The value is', factorial(6))
```

```
The value is 720
```

In [107]:

```
# Define a function that gives the total of the first ten numbers
def total_numbers(number, sum):
    if number == 11:
        return sum
    else:
        return total_numbers(number+1, sum+number)

print('The total of first ten numbers is', total_numbers(1, 0))
```

The total of first ten numbers is 55

### Nested functions

In [111]:

```
# Define a function that add a number to another number
def added_num(num1):
    def incremented_num(num1):
        num1 = num1 + 1
        return num1
    num2 = incremented_num(num1)
    print(num1, '------->>', num2)

added_num(25)
```

25 -------->> 26

### nonlocal function

In [112]:

```
# Define a function regarding 'nonlocal' function
def print_year():
    year = 1990
    def print_current_year():
        nonlocal year
        year += 32
        print('Current year is', year)
    print_current_year()
print_year()
```

Current year is 2022

In [117]:

```
# Define a function giving a message
def function(name):
    msg = 'Hi ' + name
    return msg

name = input('Enter a name: ')
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
  - IdentationError

#### ZeroDivisionError

In [1]:

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

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]:

```
nlis = []
count = 0
try:
    mean = count/len(nlis)
    print('The mean value is', mean)
except ZeroDivisionError:
    print('This code gives a ZeroDivisionError')

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]:

```
# The following code is like 1/0.
try:
    True/False
except ZeroDivisionError:
    print('The code gives a ZeroDivisionError.')

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]:

```
nlis = []
count = 0
try:
    mean = count/len(nlis)
    print('The mean value is', mean)
except ZeroDivisionError:
    print('This code gives a ZeroDivisionError')

# Since the variable 'mean' is not defined, it gives us a 'NameError
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]:

```
try:
    y = x+5
except NameError:
    print('This code gives a NameError.')

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]:

```
# Define a function giving a NameError
def addition(x, y):
    z = x + y
    return z

print('This function gives a NameError.')
total = add(3.14, 1.618)
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]:

```
# Since 'Mustafa' is not defined, the following code gives us a 'NameError.'
try:
    name = (Mustafa)
    print(name, 'today is your wedding day.')
except NameError:
    print('This code gives a NameError.')

name = (Mustafa)
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]:

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

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]:

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

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]:

```
dictionary = {'euler_constant': 0.577, 'golden_ratio': 1.618}
try:
    dictonary = dictionary['euler_number']
except KeyError:
    print('This code gives us a KeyError.')

dictonary = dictionary['euler_number']
print(dictonary)
```

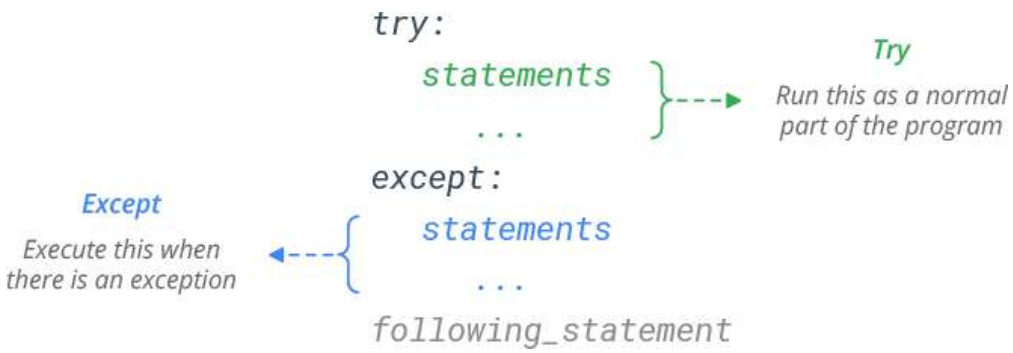
This code gives us a KeyError.

```
---------------------------------------------------------------------------
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 dictonary = dictionary['euler_number']
      8 print(dictonary)

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=NSkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkPY0101ENSkillsNetwork19487395-2021-01-01) from this connection.

## Exception Handling

### try/except




In [11]:

```
try:
    print(name)
except NameError:
    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]:

```
num1 = float(input('Enter a number:'))
print('The entered value is', num1)
try:
    num2 = float(input('Enter a number:'))
    print('The entered value is', num2)
    value = num1/num2
    print('This process is running with value = ', value)
except:
    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

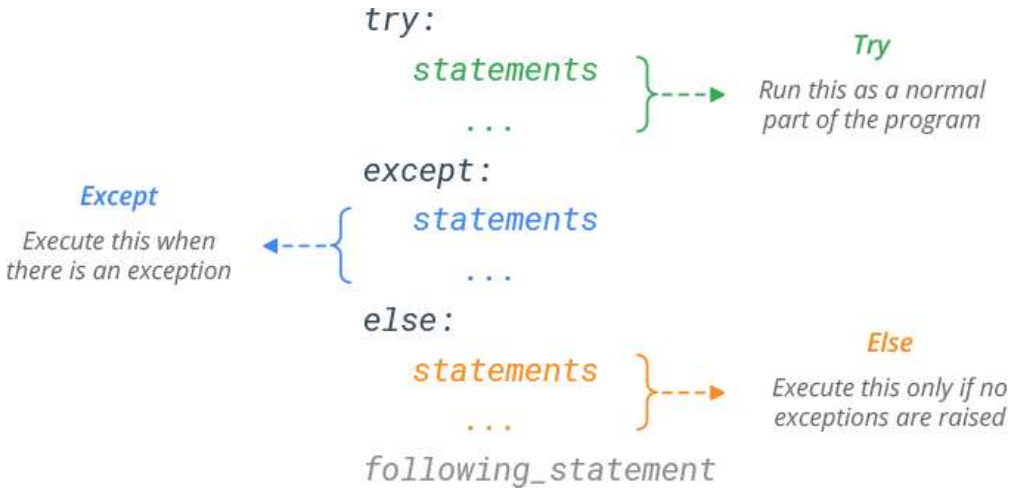
#### *try/except/except etc.*

In [2]:

```
num1 = float(input('Enter a number:'))
print('The entered value is', num1)
try:
    num2 = float(input('Enter a number:'))
    print('The entered value is', num2)
    value = num1/num2
    print('This process is running with value = ', value)
except ZeroDivisionError:
    print('This function gives a ZeroDivisionError since a number cannot divide by 0.')
except ValueError:
    print('You should provide a number.')
except:
    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*

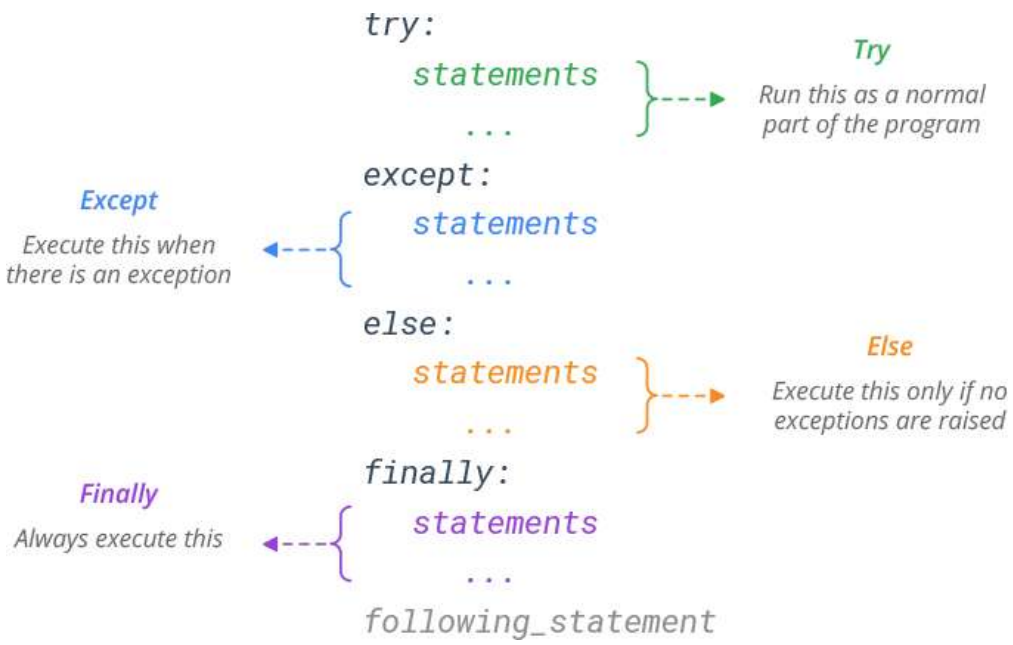


In [3]:

```
num1 = float(input('Enter a number:'))
print('The entered value is', num1)
try:
    num2 = float(input('Enter a number:'))
    print('The entered value is', num2)
    value = num1/num2
except ZeroDivisionError:
    print('This function gives a ZeroDivisionError since a number cannot divide by 0.')
except ValueError:
    print('You should provide a number.')
except:
    print('Soething went wrong!')
else:
    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]:

```
num1 = float(input('Enter a number:'))
print('The entered value is', num1)
try:
    num2 = float(input('Enter a number:'))
    print('The entered value is', num2)
    value = num1/num2
except ZeroDivisionError:
    print('This function gives a ZeroDivisionError since a number cannot divide by 0.')
except ValueError:
    print('You should provide a number.')
except:
    print('Soething went wrong!')
else:
    print('This process is running with value = ', value)
finally:
    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]:

```
num1 = float(input('Enter a number:'))
print('The entered value is', num1)
try:
    num2 = float(input('Enter a number:'))
    print('The entered value is', num2)
    value = num1/num2
except (ZeroDivisionError, NameError, ValueError):  #Multiple except clauses
    print('This function gives a ZeroDivisionError, NameError or ValueError.')
except:
    print('Soething went wrong!')
else:
    print('This process is running with value = ', value)
finally:
    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]:

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

**Built-in Functions**

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

### abs()

Returns the absolute value of a number

In [3]:

```
num1 = int(input('Enter a number: '))
print('The entered number is', num1)
num2 = float(input('Enter a number: '))
print('The entered number is', num2)
print('The absolute value of the first number is', abs(num1))
print('The absolute number of the second number is', abs(num2))
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()

Retturns *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]:

```
nlis1 = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(all(nlis1))
nlis1.append(0)    # Add '0' to the end of the list
print(nlis1)
print(all(nlis1))
nlis1.append(False)    # Adds 'False' to the end of the list
print(nlis1)
print(all(nlis1))
nlis1.clear()      # It returns an emtpy list
print(nlis1)
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 specificied integer

In [14]:

```
num = int(input('Enter a number: '))
print(f'The entered number is {num}.')
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]:

```
lis, dict, tuple = [], {}, ()
print(bool(lis), bool(dict), bool(tuple))
lis, dict, tuple = [0], {'a': 1}, (1,)
print(bool(lis), bool(dict), bool(tuple))
lis, dict, tuple = [0.0], {'a': 1.0}, (1.0,)
print(bool(lis), bool(dict), bool(tuple))
a, b, c = 0, 3.14, 'Hello, Python!'
print(bool(a), bool(b), bool(c))
statement = None
print(bool(None))
true = True
print(bool(true))
```

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

## bytes()

Returns a *btyes* object

In [26]:

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

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

## callable()

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

In [31]:

```
var = 3.14
print(callable(var))      # since the object does not appear callable, it returns False

def function():           # since the object appears callable, it returns True
    print('Hi, Python!')
msg = function
print(callable(msg))
```

```
False
True
```

## chr()

It returns a character from the specified Unicode code.

In [134]:

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

```
B
Y

ϽϽ


---------------------------------------------------------------------------
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]:

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

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

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

```
<class 'code'>
Result = 19.87
```

## exec()

Executes the specified code or object

In [39]:

```
var = 3.14
exec('print(var==3.14)')
exec('print(var!=3.14)')
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]:

```
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are special.'

special_numbers = SpecialNumbers()
print('The euler number is', getattr(special_numbers, 'euler_number'))
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]:

```
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are special.'

    def parameter(self):
        print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)

special_numbers = SpecialNumbers()
special_numbers.parameter()
delattr(SpecialNumbers, 'msg')     # The code deleted the 'msg'.
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]:

```
name = dict()
print(name)

dictionary = dict(euler_constant = 0.577, euler_number=2.718, golden_ratio=1.618)
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]:

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

```
0 Hello Python!
1 Hello, World!
```

In [155]:

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

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

## filter()

It excludes items in an iterable object.

In [159]:

```
def filtering(data):
    if data > 30:
        return data

data = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
result = filter(filtering, data)
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]:

```
nlis = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(nlis)
print(any(nlis))
nlis.clear()
print(nlis)
print(any(nlis))      # An emptly list returns False
nlis.append(0)
print(nlis)
print(any(nlis))      # 0 in a list returns False
nlis.append(False)
print(nlis)
print(any(nlis))      # False in a list returns False
nlis.append(True)
print(nlis)
print(any(nlis))      # True in a list returns True
nlis.append(1)
print(nlis)
print(any(nlis))   # 1 in a list returns True
nlis.clear()
nlis.append(None)
print(nlis)
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]:

```
txt = 'Hello, Python!'
print(ascii(txt))
text = 'Hello, Pythän!'
print(ascii(text))
print('Hello, Pyth\xe4n!')
msg = 'Hellü, World!'
print(ascii(msg))
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]:

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

```
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are special.'

special_numbers = SpecialNumbers()
print('The euler number is', hasattr(special_numbers, 'euler_number'))
print('The golden ratio is', hasattr(special_numbers, 'golden_ratio'))
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]:

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

```
322818021289917443
1330471416316301312
-7855314544920281827
1729
-6529577050584256413
```

## help()

Executes the built-in help system

In [167]:

```
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]:

```
import pandas as pd
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]:

```
print(id('Hello, Python!'))
print(id(3.14))
print(id(1729))
special_nums_list = [0.577, 1.618, 2.718, 3.14, 28, 37, 1729]
print(id(special_nums_list))
special_nums_tuple = (0.577, 1.618, 2.718, 3.14, 28, 37, 1729)
print(id(special_nums_tuple))
special_nums_set = {0.577, 1.618, 2.718, 3.14, 28, 37, 1729}
print(id(special_nums_set))
special_nums_dict = {'Euler constant': 0.577, 'Golden ratio': 1.618,
                     'Euler number': 2.718, 'PI number': 3.14,
                     'Perfect number': 28, 'Prime number': 37,
                     'Ramanujan Hardy number': 1729}
print(id(special_nums_dict))
```

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

## eval()

This function evaluates and executes an expression.

In [26]:

```
num = int(input('Enter a number: '))
print(f'The entered number is {num}.')
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]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 28, 37, 1729]
def division(number):
    return number/number

division_number_iterator = map(division, special_nums)
divided_nums = list(division_number_iterator)
print(divided_nums)

# 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]:

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

The length of the list is 8

In [59]:

```
# Calculate the average of values in the following list
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
count = 0
for i in special_nums:
    count = count + i
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]:

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

0.577

## max()

Returns the largest item in an iterable

In [171]:

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

1729

## sum()

To get the sum of numbers in a list

In [172]:

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

1808.0529999999999

## float()

It returns a floating point number.

In [28]:

```
int_num = 37
print(float(int_num))
float_num = 3.14
print(float(float_num))
txt = '2.718'
print(float(txt))
msg = 'Hello, Python!'        # It resturns a ValueError
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 resturns 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]:

```
# integer format
int_num = 37
print(format(num, 'd'))
# float numbers
float_num = 2.7182818284
print(format(float_num, 'f'))
# binary format
num = 1729
print(format(num, 'b'))
```

```
37
2.718282
11011000001
```

## hex()

Converts a number into a hexadecimal value

In [184]:

```
print(hex(6))
print(hex(37))
print(hex(1729))
```

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

## input()

Allowing user input

In [219]:

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

```
The entered message is Hello, Python!
```

## int()

Returns an integer number

In [223]:

```
num1 = int(6)
num2 = int(3.14)
num3 = int('28')
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]:

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

True

In [225]:

```
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are very special'

    def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
        self.euler_constant = euler_constant
        self.euler_number = euler_number
        self.pi = pi
        self.golden_ratio = golden_ratio
        self.msg = msg

special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are very special.')
nums = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
print(isinstance(special_numbers, SpecialNumbers))
print(isinstance(nums, SpecialNumbers))
```

True
False

## issubclass()

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

In [263]:

```
class Circle:
    def __init__(circleType):
        print('Circle is a ', circleType)

class Square(Circle):
    def __init__(self):

        Circle.__init__('square')

print(issubclass(Square, Circle))
print(issubclass(Square, list))
print(issubclass(Square, (list, Circle)))
print(issubclass(Circle, (list, Circle)))
```

```
True
False
True
True
```

## iter()

It returns an iterator object.

In [52]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
special_nums_iter = iter(special_nums)
print('Euler constant is', next(special_nums_iter))
print('The golden ratio is', next(special_nums_iter))
print('Euler number is', next(special_nums_iter))
print('Pi number is', next(special_nums_iter))
print(next(special_nums_iter), 'is a perfect number.')
print(next(special_nums_iter), 'is a perfect number.')
print(next(special_nums_iter), 'is a special and prime number.')
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]:

```
name= object()
print(type(name))
print(dir(name))
```

```
<class 'object'>
['__class__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__g
t__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__new__', '__reduce__', '__r
educe_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]:

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

```
The octal value of 37 us 0o45.
```

In [235]:

```
# decimal to octal
print('oct(1729) is:', oct(1729))

# binary to octal
print('oct(0b101) is:', oct(0b101))

# hexadecimal to octal
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]:

```
print(list())
txt = 'Python'
print(list(txt))
special_nums_set = {0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729}
print(list(special_nums_set))
special_nums_tuple = (0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729)
print(list(special_nums_tuple))
special_nums_dict = {'Euler constant': 0.577,
                     'Golden ratio': 1.618,
                     'Euler number': 2.718,
                     'Pi number': 3.14,
                     'Perfect number': 6,
                     'Prime number': 37,
                     'Ramanujan Hardy number': 1729}
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', 'Ramanu
jan Hardy number']
```

## memoryview()

It returns a memory view object.

In [91]:

```
ba = bytearray('XYZ', 'utf-8')
mv = memoryview(ba)
print(mv)
print(mv[0])
print(mv[1])
print(mv[2])
print(bytes(mv[0:2]))
print(list(mv[:]))
print(set(mv[:]))
print(tuple(mv[:]))
mv[1] = 65                  # 'Y' was replaced with 'A'
print(list(mv[:]))
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]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
number = iter(special_nums)       # Create an iteration
item = next(number)       # First item
print(item)
item = next(number)       # Second item
print(item)
item = next(number)    # Third item, etc
print(item)
item = next(number)
print(item)
item = next(number)
print(item)
item = next(number)
print(item)
item = next(number)
print(item)
item = next(number)
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 exists. |
| wb | Same as **w** but opens in binary mode. |
| w+ | Same as **w** but also alows to read from file. |
| wb+ | Same as **wb** but also alows 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]:

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

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

In [122]:

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

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

## complex()

It returns a complex number.

In [138]:

```
print(complex(1))
print(complex(2, 2))
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]:

```
name = dir()
print(name)
print()
number = 3.14
print(dir(number))
print()
nlis = [3.14]
print(dir(nlis))
print()
nset = {3.14}
print(dir(nset))
```

['FileContent', 'In', 'Out', 'SpecialNumbers', '_', '_103', '_105', '_107', '_109', '_113', '_114', '_116', '_118', '_119', '_144', '_39', '_68', '_92', '_98', '__', '___', '__builtin__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', '__vsc_ipynb_file__', '_dh', '_i', '_i1', '_i10', '_i100', '_i101', '_i102', '_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', '_i127', '_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', '_i48', '_i49', '_i5', '_i50', '_i51', '_i52', '_i53', '_i54', '_i55', '_i56', '_i57', '_i58', '_i59', '_i6', '_i60', '_i61', '_i62', '_i63', '_i64', '_i65', '_i66', '_i67', '_i68', '_i69', '_i7', '_i70', '_i71', '_i72', '_i73', '_i74', '_i75', '_i76', '_i77', '_i78', '_i79', '_i8', '_i80', '_i81', '_i82', '_i83', '_i84', '_i85', '_i86', '_i87', '_i88', '_i89', '_i9', '_i90', '_i91', '_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', 'special_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__', '__getattribute__', '__getformat__', '__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__', '__rpow__', '__rsub__', '__rtruediv__', '__set_format__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', 'as_integer_ratio', 'conjugate', 'fromhex', 'hex', 'imag', 'is_integer', 'real']

['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__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', 'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']

## divmod()

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

In [152]:

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

```
(5.0, 0.25500000000000034)
(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]:

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

```
class SpecialNumbers:
    euler_constant = 0.0
    euler_number = 0.0
    pi = 0.0
    golden_ratio = 0.0
    msg = ''

    def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
        self.euler_constant = euler_constant
        self.euler_number = euler_number
        self.pi = pi
        self.golden_ratio = golden_ratio
        self.msg = msg

special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are special.')
print(special_numbers.euler_constant)
print(special_numbers.euler_number)
print(special_numbers.pi)
print(special_numbers.golden_ratio)
print(special_numbers.msg)
setattr(special_numbers, 'Ramanujan_Hardy_number', 1729)
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]:

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

```
special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6,  37]
print(sorted(special_nums))
txt = 'Hello, Python!'
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: $\alpha^2$

In [241]:

```
print(ord('9'))
print(ord('X'))
print(ord('W'))
print(ord('^'))
```

```
57
88
87
94
```

## pow()

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

In [247]:

```
print(pow(2.718, 3.14))
print(pow(-25, -2))
print(pow(16, 3))
print(pow(-6, 2))
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]:

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

```
Hello, Python!
```

## range()

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

In [254]:

```
print(list(range(0)))
print(list(range(9)))
print(list(range(2, 9)))
for i in range(2, 9):
    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]:

```
txt = 'Python'
print(list(reversed(txt)))
special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6,  37]
print(list(reversed(special_nums)))
nums = range(6, 28)
print(list(reversed(nums)))
special_nums_tuple = (2.718, 1729, 0.577, 1.618, 28, 3.14, 6,  37)
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]:

```
print(round(3.14))
print(round(2.718))
print(round(0.577))
print(round(1.618))
print(round(1729))
```

```
3
3
1
2
1729
```

## str()

Returns the string version of the given object.

In [268]:

```
num  = 3.14
val = str(num)
print(val)
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]:

```
special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6,  37]
special_nums_tuple = tuple(special_nums)
print(special_nums_tuple)

txt = 'Hello, Python!'
txt_tuple = tuple(txt)
print(txt_tuple)

dictionary = {'A': 0.577, 'B': 2.718, 'C': 3.14}
dictionary_tuple = tuple(dictionary)
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]:

```
special_nums = [2.718, 1729, 0.577, 1.618, 28, 3.14, 6,  37]
special_nums_tuple = tuple(special_nums)
print(special_nums_tuple)
print(type(special_nums))
print()
txt = 'Hello, Python!'
txt_tuple = tuple(txt)
print(txt_tuple)
print(type(txt))
print()
dictionary = {'A': 0.577, 'B': 2.718, 'C': 3.14}
dictionary_tuple = tuple(dictionary)
print(dictionary_tuple)
print(type(dictionary))
print()
special_nums_set = {2.718, 1729, 0.577, 1.618, 28, 3.14, 6,  37}
special_nums_tuple = tuple(special_nums_set)
print(special_nums_tuple)
print(type(special_nums_set))
print()
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are very special'

    def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
        self.euler_constant = euler_constant
        self.euler_number = euler_number
        self.pi = pi
        self.golden_ratio = golden_ratio
        self.msg = msg

special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are very special.')
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]:

```
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are very special'

    def __init__(self, euler_constant, euler_number, pi, golden_ratio, msg):
        self.euler_constant = euler_constant
        self.euler_number = euler_number
        self.pi = pi
        self.golden_ratio = golden_ratio
        self.msg = msg

special_numbers = SpecialNumbers(0.577, 2.718, 3.14, 1.618, 'These numbers are very special.')
print(vars(special_numbers))
```

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

## zip()

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

In [283]:

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

```
[]
{('Pi number', 3.14), ('Perfect number', 28), ('Euler number', 2.718), ('Euler constant', 0.577), ('Ramanuja
n-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]:

```
class SpecialNumbers(object):
    def __init__(self, special_numbers):
        print('6 and 28 are', special_numbers)

class PerfectNumbers(SpecialNumbers):
    def __init__(self):

        # call superclass
        super().__init__('perfect numbers.')
        print('These numbers are very special in mathematik.')

nums = PerfectNumbers()
```

```
6 and 28 are perfect numbers.
These numbers are very special in mathematik.
```

In [294]:

```
class Animal(object):
    def __init__(self, AnimalName):
        print(AnimalName, 'lives in a farm.')

class Cow(Animal):
    def __init__(self):
        print('Cow gives us milk.')
        super().__init__('Cow')

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]:

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

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

In [304]:

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

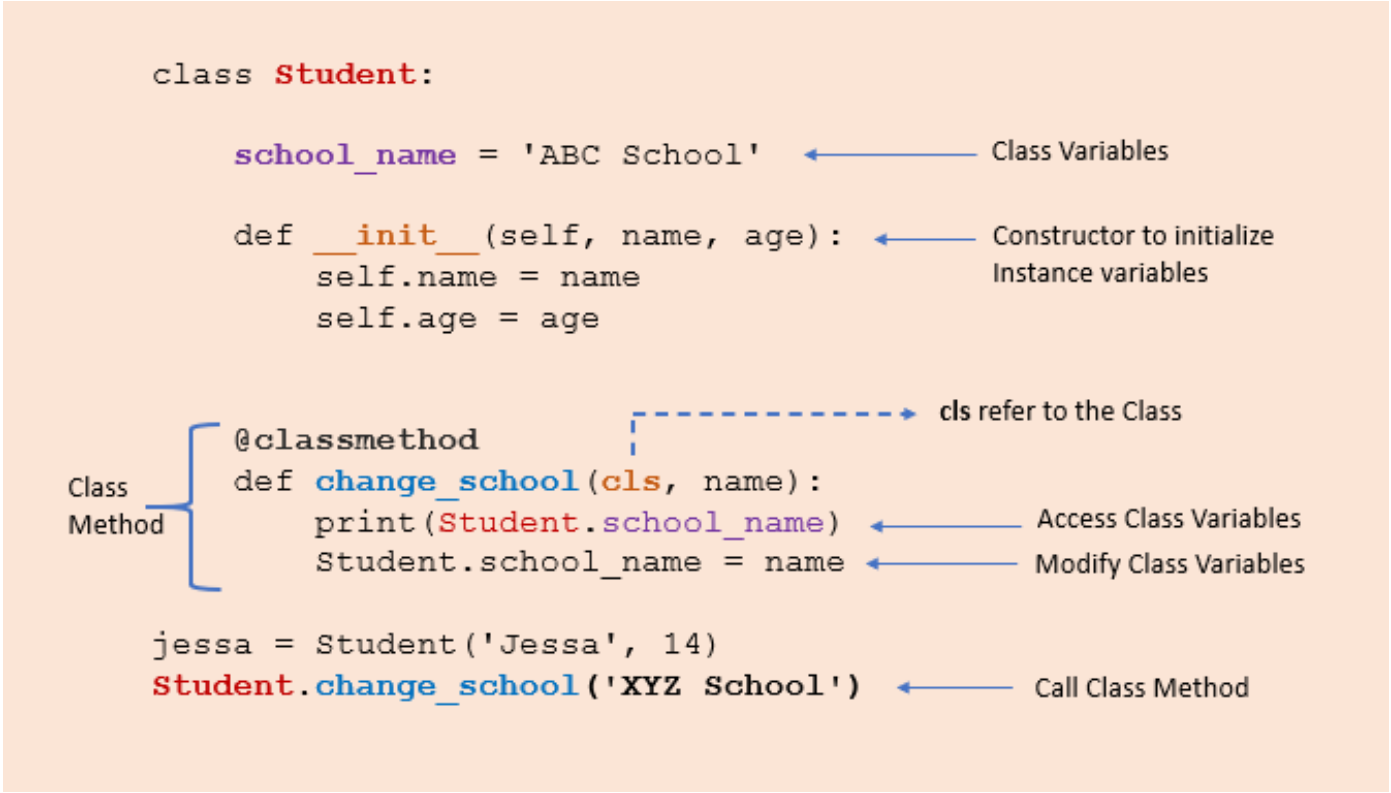
```
3.14
2.718
64.0
0.006737946999085467
0.999896315728952
720
```

# 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.




### Create a class

In [40]:

```
class Data:
  num = 3.14

print(Data)
```

```
<class '__main__.Data'>
```

### Create an object

In [41]:

```
class Data:
    num = 3.14

var = Data()
print(var.num)
```

3.14

## Function init()

In [43]:

```
class Data:
    def __init__(self, euler_number, pi_number, golden_ratio):
        self.euler_number = euler_number
        self.pi_number = pi_number
        self.golden_ratio = golden_ratio

val = Data(2.718, 3.14, 1.618)

print(val.euler_number)
print(val.golden_ratio)
print(val.pi_number)
```

2.718
1.618
3.14

## Methods

In [45]:

```
class Data:
    def __init__(self, euler_number, pi_number, golden_ratio):
        self.euler_number = euler_number
        self.pi_number = pi_number
        self.golden_ratio = golden_ratio
    def msg_function(self):
        print("The euler number is", self.euler_number)
        print("The golden ratio is", self.golden_ratio)
        print("The pi number is", self.pi_number)

val = Data(2.718, 3.14, 1.618)
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]:

```
"""
The following codes are the same as the above codes under the title 'Methods'.
You see that the output is the same, but this codes contain 'classFirstParameter' instead of 'self'.
"""
class Data:
    def __init__(classFirstParameter, euler_number, pi_number, golden_ratio):
        classFirstParameter.euler_number = euler_number
        classFirstParameter.pi_number = pi_number
        classFirstParameter.golden_ratio = golden_ratio

    def msg_function(classFirstParameter):
        print("The euler number is", classFirstParameter.euler_number)
        print("The golden ratio is", classFirstParameter.golden_ratio)
        print("The pi number is", classFirstParameter.pi_number)

val = Data(2.718, 3.14, 1.618)
val.msg_function()
```

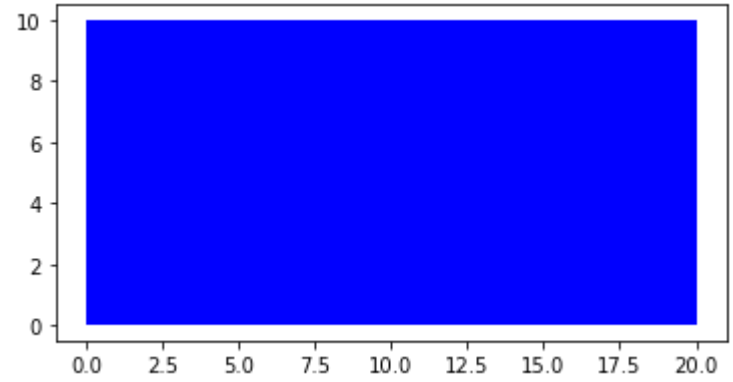
```
The euler number is 2.718
The golden ratio is 1.618
The pi number is 3.14
```

## Creating a Class to draw a Rectangle

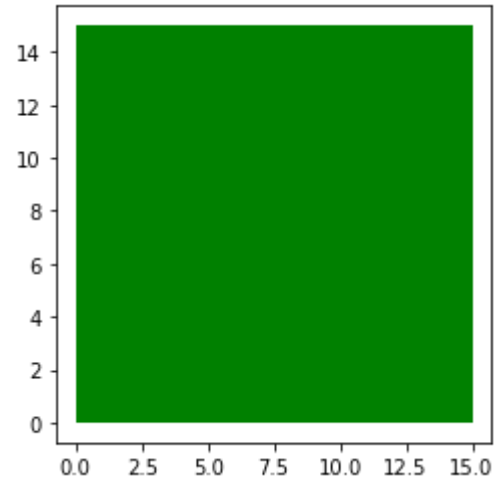
In [1]:

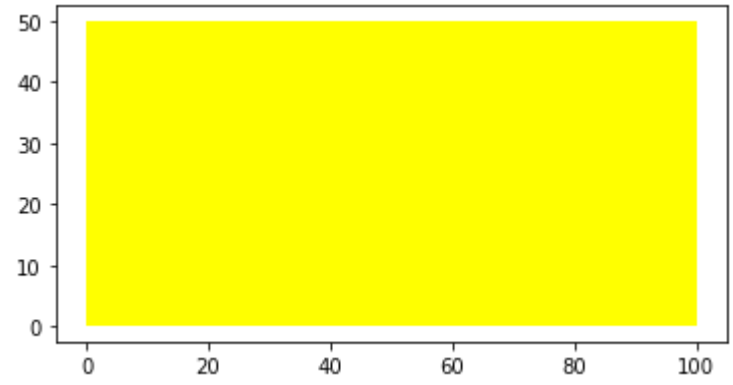
```
# Creating a class to draw a rectangle
class Rectangle(object):

    # Contructor
    def __init__(self, width, height, color):
        self.width = width
        self.height = height
        self.color = color

    # Method
    def drawRectangle(self):
        plt.gca().add_patch(plt.Rectangle((0, 0), self.width, self.height, fc=self.color))
        plt.axis('scaled')
        plt.show()

# import library to draw the Rectangle
import matplotlib.pyplot as plt
%matplotlib inline

# creating an object blue rectangle
one_Rectangle = Rectangle(20, 10, 'blue')

# Printing the object attribute width
print(one_Rectangle.width)

# Printing the object attribute height
print(one_Rectangle.height)

# Printing the object attribute color
print(one_Rectangle.color)

# Drawing the object
one_Rectangle.drawRectangle()

# Learning the methods that can be utilized on the object 'one_rectangle'
print(dir(one_Rectangle))

# We can change the properties of the rectangle
one_Rectangle.width = 15
one_Rectangle.height = 15
one_Rectangle.color = 'green'
one_Rectangle.drawRectangle()

# Using new variables, we can change the properties of the rectangle
two_Rectangle = Rectangle(100, 50, 'yellow')
two_Rectangle.drawRectangle()


```

```
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']



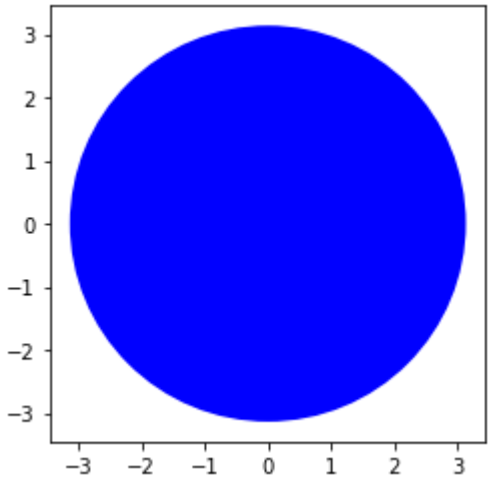

## Creating a class to draw a circle

In [3]:

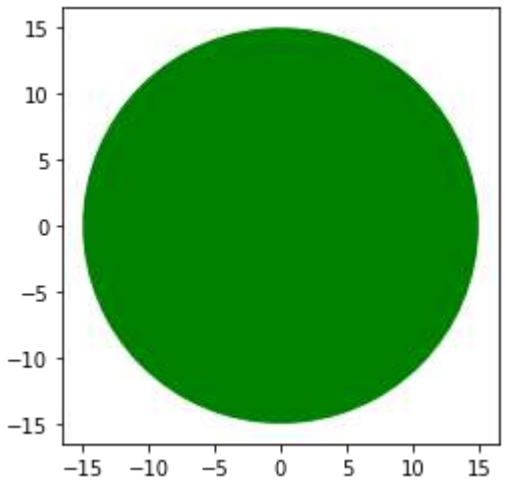
```
# Creating a class to draw a circle
class Circle(object):

    # Contructor
    def __init__(self, radius, color):
        self.radius = radius
        self.color = color

    # Method
    def increase_radius(self, r):
        self.radius = self.radius + r
        return self.radius

    # Method
    def drawCircle(self):
        plt.gca().add_patch(plt.Circle((0, 0), self.radius, fc=self.color))
        plt.axis('scaled')
        plt.show()

# import library to draw the circle
import matplotlib.pyplot as plt
%matplotlib inline

# creating an object blue circle
one_Circle = Circle(3.14, 'blue')

# Printing the object attribute radius
print(one_Circle.radius)

# Printing the object attribute color
print(one_Circle.color)

# Drawing the object
one_Circle.drawCircle()

# Learning the methods that can be utilized on the object 'one_rectangle'
print(dir(one_Circle))

# We can change the properties of the rectangle
one_Circle.radius = 15
one_Circle.color = 'green'
one_Circle.drawCircle()

# Using new variables, we can change the properties of the rectangle
two_Circle = Circle(100, 'yellow')
print(two_Circle.radius)
print(two_Circle.color)
two_Circle.drawCircle()

# Changing the radius of the object
print('Before increment: ',one_Circle.radius)
one_Circle.drawCircle()

# Increment by 15 units
one_Circle.increase_radius(15)
print('Increase the radius by 15 units: ', one_Circle.radius)
one_Circle.drawCircle()

# Increment by 30 units
```

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

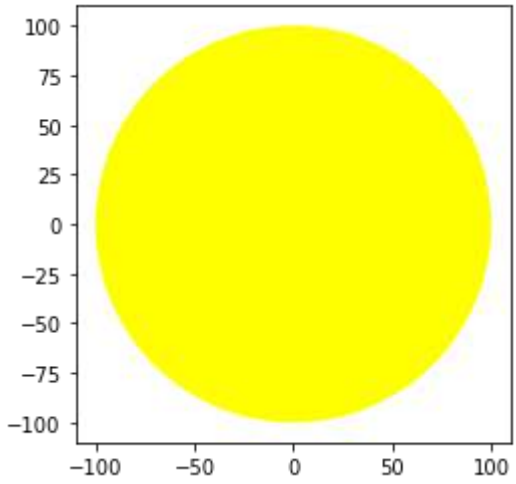
3.14
blue



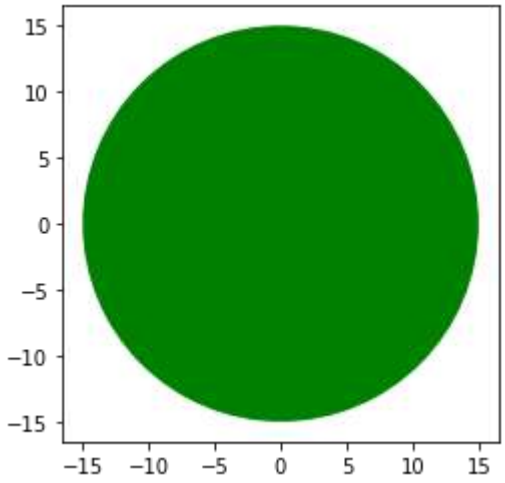
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',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



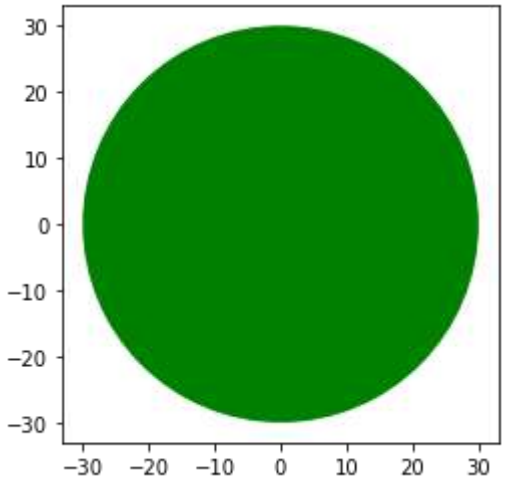
100
yellow



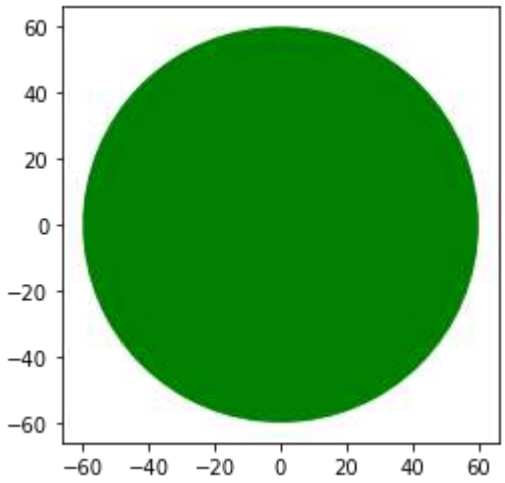

Before increment: 15



Increase the radius by 15 units:  30




Increase the radius by 30 units:  60



## Some examples

In [36]:

```
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi_number = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are special.'

special_numbers = SpecialNumbers()
print('The euler number is', getattr(special_numbers, 'euler_number'))
print('The golden ratio is', special_numbers.golden_ratio)
print('The pi number is', getattr(special_numbers, 'pi_number'))
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]:

```
class SpecialNumbers:
    euler_constant = 0.577
    euler_number = 2.718
    pi = 3.14
    golden_ratio = 1.618
    msg = 'These numbers are special.'

    def parameter(self):
        print(self.euler_constant, self.euler_number, self.pi, self.golden_ratio, self.msg)

special_numbers = SpecialNumbers()
special_numbers.parameter()
delattr(SpecialNumbers, 'msg')     # The code deleted the 'msg'.
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]:

```
class ComplexNum:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def data(self):
        print(f'{self.a}-{self.b}j')

var = ComplexNum(3.14, 1.618)
var.data()
```

```
3.14-1.618j
```

## Create a Data Classs

In [54]:

```
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

#Use the Data class to create an object, and then execute the microorganism method
value = Data('Aspergillus', 'niger')
value.microorganism()
```

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

## Create a Child Class in Data Class

In [56]:

```
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    pass

value = Recombinant('Aspergillus', 'sojae')
value.microorganism()
```

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

## Addition of init() Functions

In [4]:

```
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    def __init__(self, genus, species):
        Data.__init__(self, genus, species)

value = Recombinant('Aspergillus', 'sojae')
value.microorganism()
```

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

## Addition of super() Function

In [68]:

```
class SpecialNumbers(object):
    def __init__(self, special_numbers):
        print('6 and 28 are', special_numbers)

class PerfectNumbers(SpecialNumbers):
    def __init__(self):

        # call superclass
        super().__init__('perfect numbers.')
        print('These numbers are very special in mathematik.')

nums = PerfectNumbers()
```

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

In [71]:

```
class Animal(object):
    def __init__(self, AnimalName):
        print(AnimalName, 'lives in a farm.')

class Cow(Animal):
    def __init__(self):
        print('Cow gives us milk.')
        super().__init__('Cow')

result = Cow()
```

Cow gives us milk.
Cow lives in a farm.

In [60]:

```
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    def __init__(self, genus, species):
        super().__init__(genus, species)        # 'self' statement in this line was deleted as different from the above codes

value = Recombinant('Aspergillus', 'sojae')
value.microorganism()
```

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

## Addition of Properties under the super() Function

In [65]:

```
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    def __init__(self, genus, species):
        super().__init__(genus, species)
        self.activity = 2500        # This information was adedd as a Property

value = Recombinant('Aspergillus', 'sojae')
print(f'The enzyme activity increased to {value.activity} U/mL.')
```

```
The enzyme activity increased to 2500 U/mL.
```

In [66]:

```
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    def __init__(self, genus, species, activity):
        super().__init__(genus, species)
        self.activity = activity        # This information was adedd as a Property

value = Recombinant('Aspergillus', 'sojae', 2500)
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]:

```
class Data:
    def __init__(self, genus, species):
        self.genus = genus
        self.species = species

    def microorganism(self):
        print(f'The name of a microorganism is in the form of {self.genus} {self.species}.')

class Recombinant(Data):
    def __init__(self, genus, species, activity):
        super().__init__(genus, species)
        self.activity = activity        # This information was adedd as a Property

    def increment(self):
        print(f'With this new recombinant {self.genus} {self.species} strain, the enzyme activity increased 2-times with {se

value = Recombinant('Aspergillus', 'sojae', 2500)
value.increment()
```

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

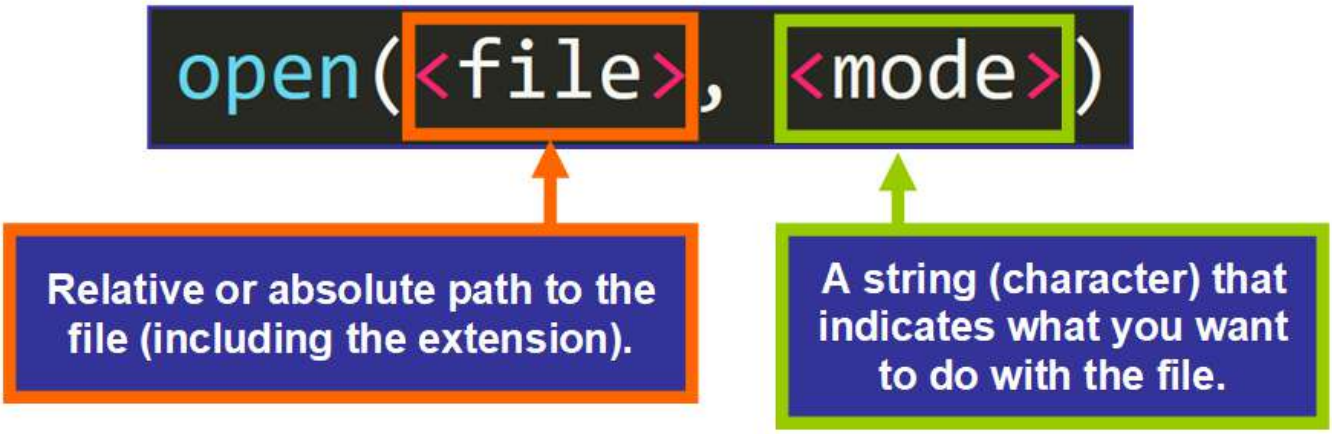
# Python Tutorial

**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 |
|---|---|
| writeable() | Returns whether the file can be written to or not |
| readable() | Returns whether the file stream can be read or not |
| read() | Returns the file content |
| readline() | Returns one line from the file |
| readlines() | Returns a list of lines from the file |
| write() | Writes the specified string to the file |
| writelines() | Writes a list of strings to the file |

| Method | Description |
|---|---|
| close() | Closes the file |
| flush() | Flushes the internal buffer |
| seek() | Change the file position |
| tell() | Returns the current file |
| truncate() | Resizes the file to a specified |

### Reading file

In [2]:

```
# Reading the txt file
file_name = "pcr_file.txt"
file = open(file_name, "r")
content = file.read()
content
```

Out[2]:

'I dedicate this book to Nancy Lier Cosgrove Mullis.\nJean-Paul Sartre somewhere observed that we eac
h 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 peopl
e around him. She will be his morning and his evening star, shining with the brightest and the softest lig
ht in his heaven. She will be the end of his wanderings, and their love will arouse the daffodils in the spri
ng 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.\nOr maybe he would have just said, “If I’d had a woman
like that, my books would not have been about despair.”\nThis book is not about despair. It is about a litt
le 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 de
pth; it is due to a year of Nancy, and the prospect of never again being without her.\n\n'

In [3]:

```
# Printing the path of file
print(file.name)
# Printing the mode of file
print(file.mode)
# Printing the file with '\n' as a new file
print(content)
# Printing the type of file
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 u
s. 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 f
aith 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 d
espair.”
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 neve
r again being without her.


<class 'str'>

In [4]:

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

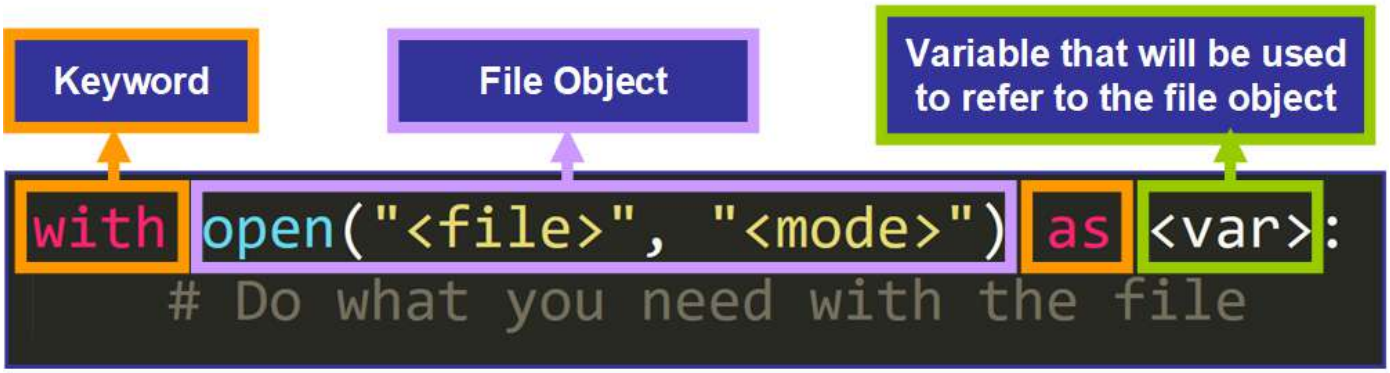
In [5]:

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

Out[5]:

True

## Another way to read a file




In [6]:

```
fname = 'pcr_file.txt'
with open(fname, 'r') as f:
    content = f.read()
    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 u s. 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 f aith 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 d espair.”
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 neve r again being without her.

In [7]:

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

Out[7]:

True

In [8]:

```
# See the content of the file
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 u
s. 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 f
aith 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 d
espair.”
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 neve
r again being without her.
```

In [9]:

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

```
I dedicate this book
```

In [10]:

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

```
I dedicate this book
 to Nancy Lier Cosgr
ove Mullis.
Jean-Paul Sartre somewhere observed th
at we each of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may h
a
```

In [11]:

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

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

In [12]:

```
# Difference between read() and readline()
with open(fname, 'r') as f:
    print(f.readline(20))
    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]:

```
with open(fname, 'r') as f:
    line_number = 1
    for line in f:
        print('Line number', str(line_number), ':', line)
        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, mig
ht get very lucky, and make his own heaven out of one of the people around him. She will be his mornin
g 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 preced
e the irises. Their faith in one another will be deeper than time and their eternal spirit will be seamless o
nce again.

Line number 3 : Or maybe he would have just said, “If I’d had a woman like that, my books would not ha
ve 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 pr
ospect of never again being without her.

Line number 5 :
```

## Methods

### read(n) function

- Reads atmost **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]:

```
with open(fname, 'r') as f:
    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 u
s. 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 f
aith 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 d
espair.”
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 neve
r again being without her.
```

In [15]:

```
with open(fname, 'r') as f:
    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]:

```
with open(fname, 'r') as f:
    file_list = f.readlines()
    # Printing the first line
    print(file_list[0])
    # Printing the second line
    print(file_list[1])
    # Printing the third line
    print(file_list[2])
    # Printing the fourth line
    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 u
s. 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 f
aith 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 d
espair.”

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 neve
r again being without her.

### readlines() function

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

In [17]:

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

['I dedicate this book to Nancy Lier Cosgrove Mullis.\n', 'Jean-Paul Sartre somewhere observed that we e
ach of us make our own hell out of the people around us. Had Jean-Paul known Nancy, he may have not
ed that at least one man, someday, might get very lucky, and make his own heaven out of one of the pe
ople 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 s
pring to follow the crocuses and precede the irises. Their faith in one another will be deeper than time a
nd their eternal spirit will be seamless once again.\n', 'Or maybe he would have just said, “If I’d had a wo
man like that, my books would not have been about despair.”\n', 'This book is not about despair. It is ab
out 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 la
ck 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]:

```
with open(fname, 'r') as f:
    len_file = 0
    total_len_file = 0
    for line in f:
        # Total length of line in the text file
        total_len_file = total_len_file+len(line)

        # Lenght of the line after removing leading and trailing spaces
        len_file = len_file+len(line.strip())
    print(f'Total lenght of the line is {total_len_file}.')
    print(f'The length of the line after removing leading and trailing spaces is {len_file}.')

```

Total lenght 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]:

```
with open(fname, 'r') as f:
    str = ""
    for line in f:
        str+=line
    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]:

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

The number of lines in the text file is 5.

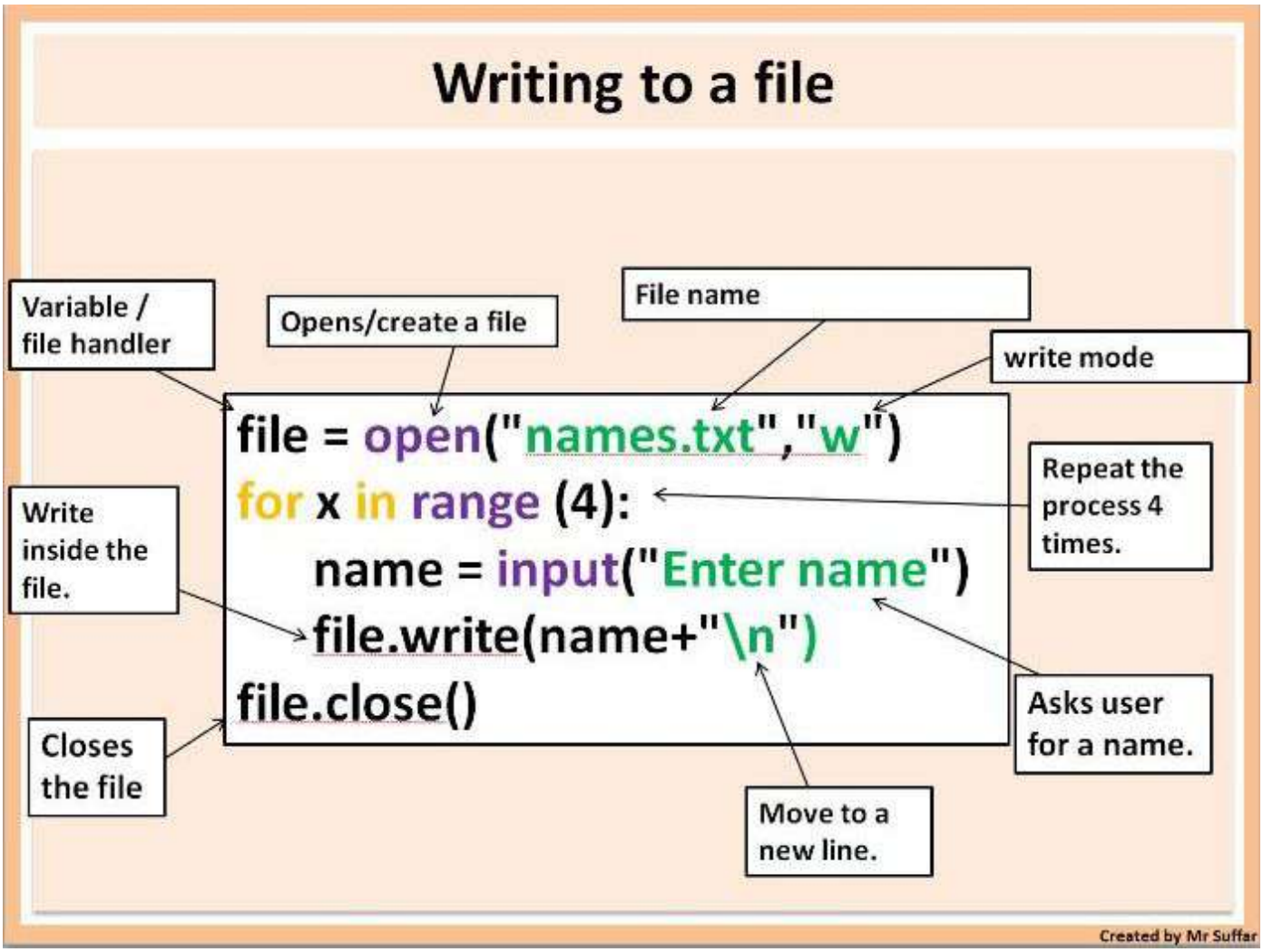
# Python Tutorial

**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 exists. |
| wb | Same as **w** but opens in binary mode. |
| w+ | Same as **w** but also alows to read from file. |
| wb+ | Same as **wb** but also alows 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. |

## Writing files

In [17]:

```
# Writing lines to a file.
fname = 'pcr_file.txt'
with open(fname, 'w') as f:
    f.write("I dedicate this book to Nancy Lier Cosgrove Mullis.\n")
    f.write("Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. H
    f.write("Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about
    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
    f.write("A feedback from Elle on the book\n")
    f.write("This bona-fide wild card of the scientific community writes with eccentric gusto.... Mullis has created a free
```

In [18]:

```
# Checking the file whether it was written or not
with open(fname, 'r') as f:
    content = f.read()
    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 u s. 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 f aith 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 abo ut 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 neve r 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 fr ee-wheeling adventure yarn that just happens to be the story of his life.

In [20]:

```
# Writing a list to a file
added_text = ["From The New Yorker\n",
                  "Entertaini ng ... [Mullis is] usefully cranky and combative, raising provocative questions about receive
                  "From Chicago Sun-Times\n",
                  "One of the most unusual scientists of our times, a man who would be a joy to put under a microscop
                  "From Andrew Weil, M.D.\n",
                  "In this entertaining romp through diverse fields of inquiry, [Mullis] displays the openmindedness, ecc
                ]

fname = 'sample.txt'
with open(fname, 'w') as f:
    for line in added_text:
        print(line)
        f.write(line)
```

From The New Yorker

Entertaini ng ... [Mullis is] usefully cranky and combative, raising provocative questions about received tr uths 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, eccent ricity, brilliance, and general curmudgeonliness that make him the colorful chracter he is. His stories are engaging, informative, and fun.

## Appending files

In [24]:

```
# Wrting and tehn reading the file
new_file = 'pcr_file.txt'
with open(new_file, 'w') as f:
    f.write('Overright\n')
with open(new_file, 'r') as f:
    print(f.read())
```

```
Overright
```

In [25]:

```
# Writing a new line to the text file
with open(new_file, 'a') as f:        # To append a new line to the text file, use 'a' in the syntax string
    f.write("I dedicate this book to Nancy Lier Cosgrove Mullis.\n")
    f.write("Jean-Paul Sartre somewhere observed that we each of us make our own hell out of the people around us. H
    f.write("Or maybe he would have just said, 'If I would had a woman like that, my books would not have been about
    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
    f.write("A feedback from Elle on the book\n")
    f.write("This bona-fide wild card of the scientific community writes with eccentric gusto…. Mullis has created a free

# Verification of the new lines in the text file
with open(new_file, 'r') as f:
    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 u
s. 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 f
aith 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 abo
ut 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 neve
r 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 fr
ee-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]:

```
fname = 'pcr_file.txt'
with open(fname, 'a+') as f:
    f.write("From F. Lee Bailey\n")
    f.write("A very good book by a fascinating man.... [Mullis] enjoys an almost frighteningly brilliant mind and yet some
    print(f.read())
```

In [9]:

```
# To verify the text file whether it is added or not
with open(fname, 'r') as f:
    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 u
s. 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 f
aith 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 abo
ut 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 neve
r 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 fr
ee-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 so
mehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quic
k, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter mo
ments aside, what he has to say is important.
```

**tell() and seek() functions with a+**

In [10]:

```
with open(fname, 'a+') as f:
    print("First location: {}".format(f.tell()))      # it returns the current position in bytes

    content = f.read()
    if not content:
        print('Read nothing.')
    else:
        print(f.read())

    f.seek(0, 0)
    """
    seek() function is used to change the position of the File Handle to a given specific position.
    File handle is like a cursor, which defines from where the data has to be read or written in the file.
    Syntax: f.seek(offset, from_what), where f is file pointer
    Parameters:
    Offset: Number of positions to move forward
    from_what: It defines point of reference.
    Returns: Return the new absolute position.
    The reference point is selected by the from_what argument. It accepts three values:
            0: sets the reference point at the beginning of the file
            1: sets the reference point at the current file position
            2: sets the reference point at the end of the file
    """
    print('\nSecond location: {}'.format(f.tell()))
    content = f.read()
    if not content:
        print('Read nothing.')
    else:
        print(content)
    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 u
s. 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 f
aith 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 abo
ut 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 neve
r 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 fr
ee-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 so
mehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quic
k, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter mo
ments aside, what he has to say is important.

Location after reading: 1641
```

**r+**

Reading and writing. Cannot truncate the file.

In [13]:

```
with open(fname, 'r+') as f:
    content=f.readlines()
    f.seek(0,0)       # writing at the beginning of the file
    f.write('From The San Diego Union-Tribune' + '\n')
    f.write("Refreshing … brashly confident … indisputably entertaining." + "\n")
    f.write("To my family..." + '\n')
    f.seek(0,0)
    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 aro
und 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 abo
ut 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 neve
r 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 fr
ee-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 so
mehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quic
k, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter mo
ments aside, what he has to say is important.
```

## Copy the file

In [14]:

```
# Let's copy the text file 'pcr_file.txt' to another one 'pcr_file_1.txt'
fname = 'pcr_file.txt'
with open(fname, 'r') as f_reading:
    with open('pcr_file_1.txt', 'w') as f_writing:
        for line in f_reading:
            f_writing.write(line)
```

In [15]:

```
# For the verification, execute the following codes
fname = 'pcr_file_1.txt'
with open(fname, 'r') as f:
    print(f.read())

# 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 aro und 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 abo ut 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 neve r 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 fr ee-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 so mehow manages to keep his feet firmly on the ground.... This guy cuts through the nonsense to the quic k, tells it like it is, and manages to do so with insouciance [and] occasional puckishness.... But lighter mo ments aside, what he has to say is important.

## Some examples

In [36]:

```
# Writing the student names into a file
fname = open(r'student_name.txt', 'w')
for i in range(3):
        name = input('Enter a student name: ')
        fname.write(name)
        fname.write('\n')           # To write names as a new line
fname = open(r'student_name.txt', 'r')
for line in fname:
    print(line)
fname.close()
```

Daniela

Axel

Leonardo

In [40]:

```
# To write the file
fname = open(r'student_name.txt', 'w')
name_list = []
for i in range(3):
    name = input('Enter a student name: ')
    name_list.append(name + '\n')
fname.writelines(name_list)

# To read the file
fname = open(r'student_name.txt', 'r')
for line in fname:
    print(line)
fname.close()
```

```
Daniel

Axel

Leonardo
```

In [41]:

```
lines = ['Hello, World!', 'Hi, Python!']
with open('new_file.txt', 'w') as f:
    for line in lines:
        f.write(line)
        f.write('\n')

with open('new_file.txt', 'r') as f:
    print(f.read())
```

```
Hello, World!
Hi, Python!
```

In [43]:

```
# Add more lines into the file
more_lines = ['Hi, Sun!', 'Hello, Summer!', 'Hi, See!']
with open('new_file.txt', 'a') as f:
    f.writelines('\n' .join(more_lines))

with open('new_file.txt', 'r') as f:
    print(f.read())

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** | >>>a="HELLO"<br>>>>print(a[0])<br>>>>H<br>>>>print(a[-1])<br>>>>O | ❖ Positive indexing helps in accessing the string from the beginning<br>❖ Negative subscript helps in accessing the string from the end. |
| | | |
| **Slicing:** | Print[0:4] – HELL<br>Print[ :3] – HEL<br>Print[0: ]- HELLO | The Slice[start : stop] operator extracts sub string from the strings.<br>A segment of a string is called a slice. |
| **Concatenation** | a="save"<br>b="earth"<br>>>>print(a+b)<br>saveearth | The + operator joins the text on both sides of the operator. |
| **Repetitions:** | a="panimalar "<br>>>>print(3*a)<br>panimalarpanimalar<br>panimalar | The * operator repeats the string on the left hand side times the value on right hand side. |
| **Membership:** | >>> s="good morning"<br>>>>"m" in s<br>True<br>>>> "a" not in s<br>True | Using membership operators to check a particular character is in string or not. Returns true if present |

### Strings are unchangeable

In [1]:

```
string_hello = 'Hello'
string_python = 'Python!'
print(string_hello)
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]:

```
text = 'Python is a programming language.'
print(text)
del text[1]
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]:

```
text = 'Python is a programming language.'
print(text)
del text
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
```

## Concetanation of strings

It combines two or more strings using the sign '+' to form a new string.

In [9]:

```
text1 = 'Hello '
text2 = 'Python!'
new_text = text1 + text2
print(new_text)
```

Hello Python!

## Appending (+=) adds a new string the the end of the current string

In [10]:

```
text1 = 'Hello '
text2 = 'Python!'
text1+=text2
print(text1)
```

Hello Python!

## To repeat a string, the multiplication (*) operator is used

In [12]:

```
text = 'Hello, Python! '
print(text*4)
```

Hello, Python! Hello, Python! Hello, Python! Hello, Python!

## Accessing the item by indexing

In [21]:

```
text = 'Hello, Python!'
print(text[0:5])      # Pozitive indexing
print(text[4])
print(text[-7:])      # Negative indexing
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]:

```
text = 'Hello, Python!'
print(len(text))
print(text[:14])      # Default stride value is 1.
print(text[0:14:2])
print(text[::3])
```

```
14
Hello, Python!
Hlo yhn
Hl tn
```

## Reverse string

The stride value is equal to -1 if a reverse string is wanted to obtain

In [30]:

```
text = 'Hello, Python!'
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]:

```
text = 'Hello, Python!'
print('H' in text)
print('H' not in text)
print('c' not in text)
print('c' in text)
```

```
True
False
True
False
```

## String Functions in Python

You can find some useful functions from the below table.

| Method | Description |
|---|---|
| capitalize() | Capitalizes first letter of string |
| find(str, beg=0 end=len(string)) | Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise |
| isalnum() | Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise |
| isalpha() | Returns true if string has at least 1 character and all characters are alphabetic and false otherwise |
| isdigit() | Returns true if string contains only digits and false otherwise |
| islower() | Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise |
| isnumeric() | Returns true if a unicode string contains only numeric characters and false otherwise |
| isspace() | Returns true if string contains only whitespace characters and false otherwise |
| istitle() | Returns true if string is properly "titlecased" and false otherwise |
| isupper() | Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise |
| join(seq) | Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string |
| len(string) | Returns the length of the string |
| lower() | Converts all uppercase letters in string to lowercase |
| max(str) | Returns the max alphabetical character from the string str |
| min(str) | Returns the min alphabetical character from the string str |
| replace(old, new [, max]) | Replaces all occurrences of old in string with new or at most max occurrences if max given |
| split(str="", num=string.count(str)) | Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given |
| splitlines( num=string.count('\n')) | Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed |
| startswith(str, beg=0,end=len(string)) | Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise |
| strip([chars]) | Performs both lstrip() and rstrip() on string |
| swapcase() | Inverts case for all letters in string |
| title() | Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase |
| upper() | Converts lowercase letters in string to uppercase |
| isdecimal() | 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]:

```
text = 'Hello, Python!'
print(f'Before casefold: {text}')
text = text.casefold()
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]:

```
text = 'Hello, Python!'
print(f'Before center() function: {text}')
text = text.center(50)
print(f'After center() function: {text}')
new_text = 'Hi, Python!'
new_text = new_text.center(50, '-')
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]:

```
text = 'Hello, Python!'
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]:

```
text = 'Hello, Python!'
text = text.endswith('Python!')
print(text)
new_text = 'Hi, Python!'
new_text = new_text.endswith('World!')
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]:

```
text = 'Hello, Python!'
print(text.find('Python'))
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]:

```
text = 'Hello {} and Hi {}'.format('World!', 'Python!')
print(text)
text = 'Hello {world} and Hi {python}'.format(world='World!', python='Python!')
print(text)
text = 'Hello {0} and Hi {1}'.format('World!', 'Python!')
print(text)
text = 'Hello {1} and Hi {0}'.format('World!', 'Python!')
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]:

```
text = 'Hello, Python!'
print(text.index('Python!'))
print(text.index('Hello'))
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]:

```
text = 'Hello, Python!'
print(text.isalnum())
msg = 'Hello1358'
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]:

```
text = 'Hello'
print(text.isalpha())
text = 'Hello1358'       # The text contains numbers.
print(text.isalpha())
text = 'Hello Python!'    # The text contains a white space.
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]:

```
text = 'Hello'
print(text.isdecimal())
numbered_text = '011235813'
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]:

```
numbered_text = '011235813'
print(numbered_text.isdigit())
```

```
True
```

## isidentifier() function

It returns *True* if the string is a valid identifier, on the contrary *False*.

In [85]:

```
numbered_text = '011235813'
print(numbered_text.isidentifier())
variable = 'numbered_text'
print(variable.isidentifier())
```

```
False
True
```

## isprintable() function

It returns *True* if all the characters in the string are printable features.

In [89]:

```
text = 'Hello, Python!'
print(text.isprintable())
new_text = 'Hello, \n Python!'
print(new_text.isprintable())
space = ' '
print(space.isprintable())
```

```
True
False
True
```

## isspace() function

It returns *True* if all the characters in the string are whitespaces.

In [90]:

```
text = 'Hello, Python!'
print(text.isspace())
space = ' '
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]:

```
text = 'Hello, Python!'
print(text.islower())
text = text.lower()          # It converts to lower case all the characters in the string.
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]:

```
text = 'Hello, Python!'
print(text.isupper())
text = text.upper()          # It converts to upper case all the characters in the string.
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]:

```
text_list = ['Hello', 'World', 'Hi', 'Python']
print('#'. join(text_list))
text_tuple =  ('Hello', 'World', 'Hi', 'Python')
print('+'. join(text_tuple))
text_set =  {'Hello', 'World', 'Hi', 'Python'}
print('--'. join(text_set))
text_dict =  {'val1': 'Hello', 'val2': 'World', 'val3': 'Hi', 'val4': 'Python'}
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]:

```
text = 'Python'
text = text.ljust(30, '-')
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]:

```
text = 'Python'
text = text.rjust(30, '-')
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]:

```
text = '          Hello Python!          '
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]:

```
text = '          Hello Python!          '
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]:

```
text = '          Hello Python!          '
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]:

```
text = 'JavaScript is a programming language.'
print(text)
modified_text = text.replace('JavaScript', 'Python', 1)
print(modified_text)
```

```
JavaScript is a programming language.
Python is a programming language.
```

In [107]:

```
text = 'Jython is a programming language.'
print(text)
modified_text = text.replace('J', 'P')
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]:

```
text = 'Hello World!, Hi Python!'
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]:

```
text = 'Hello, Python is my favorite programming language.'
print(f"'Python' is in the position {text.rfind('Python')}.")
print(f"'my' is in the position {text.rfind('my')}.")
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]:

```
text = 'Hello, Python is my favorite programming language.'
print(f"'Python' is in the position {text.rindex('Python')}.")
print(f"'my' is in the position {text.rindex('my')}.")
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]:

```
text = 'Hello Python!'
print(text.swapcase())
text = 'hELLO pYTHON!'
print(text.swapcase())
```

```
hELLO pYTHON!
Hello Python!
```

## title() function

This function converts the first character in the given string into uppercase.

In [117]:

```
text = 'hello world, hi python!'
print(text.title())
```

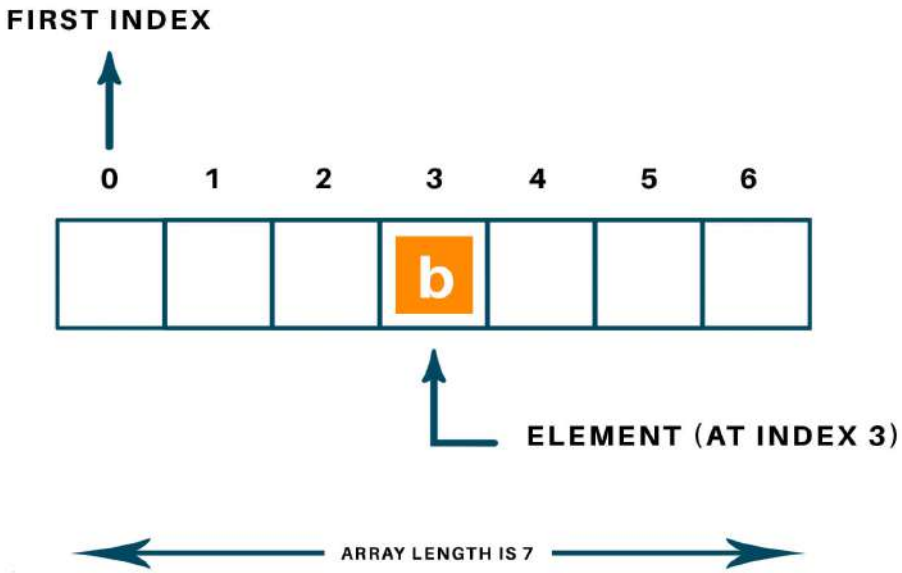
```
Hello World, Hi Python!
```

# 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]:

```
# Import the module
import array as arr
from array import *
```

In [5]:

```
# To access more information regarding array, you can execute the following commands
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]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
for i in special_nums:
    print(i)
```

```
0.577
1.618
2.718
3.14
6.0
37.0
1729.0
```

## Accessing

In [7]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
print(f'First element of numbers is {special_nums[0]}, called euler constant.')
print(f'Second element of numbers is {special_nums[1]}, called golden_ratio.')
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]:

```
nums = arr.array('i', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34])

# Changing the first element of the array
nums[0] = 55
print(nums)

# Changing 2nd to 4th elements of the array
nums[1:4] =arr.array('i', [89, 144, 233, 377])
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]:

```
nums = arr.array('i', [0, 1, 1, 2, 3, 5, 8, 13, 21, 34])

# Deleting the first element of the array
del nums[0]
print(nums)

# Deleting the 2nd to 4th elements of the array
del nums[1:4]
print(nums)
```

```
array('i', [1, 1, 2, 3, 5, 8, 13, 21, 34])
array('i', [1, 5, 8, 13, 21, 34])
```

## Lenght of the array

In [10]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
print(f'The length of the array is {len(special_nums)}.')
```

```
The length of the array is 7.
```

## Concatenation

In [11]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
fibonacci_nums = arr.array('d', [1, 1, 2, 3, 5, 8, 13, 21, 34])
special_fibonacci_nums = arr.array('d')
special_fibonacci_nums = special_nums + fibonacci_nums
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]:

```
mult = 10
one_array = [1]*mult
print(one_array)
```

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

In [13]:

```
mult = 10
nums_array = [i for i in range(mult)]
print(nums_array)
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

## Addition with the functions insert() and append()

In [14]:

```
# Using the function 'insert()'
fibonacci_nums = arr.array('i', [1, 1, 2, 3, 5, 8, 13, 21, 34])
print('Before any additon into fibonacci numbers')
for i in fibonacci_nums:
    print(i, end = ' ')
print()
print('After an element additon into fibonacci numbers')
added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
fibonacci_nums.insert(9, added_num)
for i in fibonacci_nums:
    print(i, end = ' ')
print()
print('After an element additon into fibonacci numbers')
added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
fibonacci_nums.insert(10, added_num)
for i in fibonacci_nums:
    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]:

```
# Using the function 'append()'
fibonacci_nums = arr.array('i', [1, 1, 2, 3, 5, 8, 13, 21, 34])
print('Before any additon into fibonacci numbers')
for i in fibonacci_nums:
    print(i, end = ' ')
print()
print('After an element additon into fibonacci numbers')
added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
fibonacci_nums.append(added_num)
for i in fibonacci_nums:
    print(i, end = ' ')
print()
print('After an element additon into fibonacci numbers')
added_num = fibonacci_nums[-1] + fibonacci_nums[-2]
fibonacci_nums.append(added_num)
for i in fibonacci_nums:
    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]:

```
# Using the function 'remove()'
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
print('Before removing an element from the array')
for i in special_nums:
    print(i, end = ' ')
print()
print('After removing an element from the array')
special_nums.remove(0.577)
for i in special_nums:
    print(i, end=' ')
print()
print('After removing one more element from the array')
special_nums.remove(special_nums[0])     # We can make this using indexing
for i in special_nums:
    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 [17]:

```
# Using the function 'pop()'
# The function pop() removes the last element from the array
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
print('Before removing an element from the array')
for i in special_nums:
    print(i, end = ' ')
print()
print('After removing last element from the array')
special_nums.pop()
for i in special_nums:
    print(i, end=' ')
print()
print('After removing one more last element from the array')
special_nums.pop()
for i in special_nums:
    print(i, end=' ')
print()
print('After removing one more element using index from the array')
special_nums.pop(3)        # It deleted the pi number
for i in special_nums:
    print(i, end=' ')
print()
```

```
Before removing an element from the array
0.577 1.618 2.718 3.14 6.0 37.0 1729.0
After removing last element from the array
0.577 1.618 2.718 3.14 6.0 37.0
After removing one more last element from the array
0.577 1.618 2.718 3.14 6.0
After removing one more element using index from the array
0.577 1.618 2.718 6.0
```

## Slicing

In [18]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
sliced_special_nums = special_nums[1:5]    # It returns between index 1 and index 4, not index 5.
print(sliced_special_nums)
# or using for loop
for i in sliced_special_nums:
    print(i, end = " ")
```

```
array('d', [1.618, 2.718, 3.14, 6.0])
1.618 2.718 3.14 6.0
```

In [19]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
sliced_special_nums = special_nums[3:]        # It returns index 3 and later.
print(sliced_special_nums)
```

```
array('d', [3.14, 6.0, 37.0, 1729.0])
```

In [20]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
sliced_special_nums = special_nums[:3]        # It returns until index 2, not index 3.
print(sliced_special_nums)
```

```
array('d', [0.577, 1.618, 2.718])
```

In [21]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
sliced_special_nums = special_nums[:]      # It returns all elements in the array
print(sliced_special_nums)
```

```
array('d', [0.577, 1.618, 2.718, 3.14, 6.0, 37.0, 1729.0])
```

In [22]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
sliced_special_nums = special_nums[::-1]       # It reverses the array.
print(sliced_special_nums)
```

```
array('d', [1729.0, 37.0, 6.0, 3.14, 2.718, 1.618, 0.577])
```

## Searching

In [23]:

```
# To make a search in an array, use the function index()
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
searched_item = special_nums.index(2.718)
print(f'The searched item euler number 2.718 is present at index {searched_item}.')
# printing with format
print('The searched item euler number {} is present at index {}.'.format(2.718, searched_item))
```

```
The searched item euler number 2.718 is present at index 2.
The searched item euler number 2.718 is present at index 2.
```

## Copying

### Copying using assignment

This process gives the same ID number.

In [24]:

```
special_nums = arr.array('d', [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
copied_special_nums = special_nums
print(special_nums, 'with the ID number', id(special_nums))
print(copied_special_nums, 'with the ID number', id(copied_special_nums))

# Using for loop
for i in special_nums:
    print(i, end= ' ')
print()
print(f'The ID number of the array special_nums is {id(special_nums)}.')
for i in copied_special_nums:
    print(i, end= ' ')
print()
print(f'The ID number of the array copied_special_nums is {id(copied_special_nums)}.')
```

```
array('d', [0.577, 1.618, 2.718, 3.14, 6.0, 37.0, 1729.0]) with the ID number 2668250199472
array('d', [0.577, 1.618, 2.718, 3.14, 6.0, 37.0, 1729.0]) with the ID number 2668250199472
0.577 1.618 2.718 3.14 6.0 37.0 1729.0
The ID number of the array special_nums is 2668250199472.
0.577 1.618 2.718 3.14 6.0 37.0 1729.0
The ID number of the array copied_special_nums is 2668250199472.
```

### Copying using view()

This process gives the different ID number.

In [25]:

```
# import numpy library
import numpy as np

# Copying the array
special_nums = np.array( [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
copied_special_nums = special_nums.view()
print(special_nums, 'with the ID number', id(special_nums))
print(copied_special_nums, 'with the ID number', id(copied_special_nums))

#Using for loop
for i in special_nums:
    print(i, end = ' ')
print()
for i in copied_special_nums:
    print(i, end = ' ')

```

```
[5.770e-01 1.618e+00 2.718e+00 3.140e+00 6.000e+00 3.700e+01 1.729e+03] with the ID number 2668
248532144
[5.770e-01 1.618e+00 2.718e+00 3.140e+00 6.000e+00 3.700e+01 1.729e+03] with the ID number 2668
254732400
0.577 1.618 2.718 3.14 6.0 37.0 1729.0
0.577 1.618 2.718 3.14 6.0 37.0 1729.0
```

### Copying using copy()

This process gives the different ID number.

In [26]:

```
# import numpy library
import numpy as np

# Copying the array
special_nums = np.array( [0.577, 1.618, 2.718, 3.14, 6, 37, 1729])
copied_special_nums = special_nums.copy()
print(special_nums, 'with the ID number', id(special_nums))
print(copied_special_nums, 'with the ID number', id(copied_special_nums))

#Using for loop
for i in special_nums:
    print(i, end = ' ')
print()
for i in copied_special_nums:
    print(i, end = ' ')

```

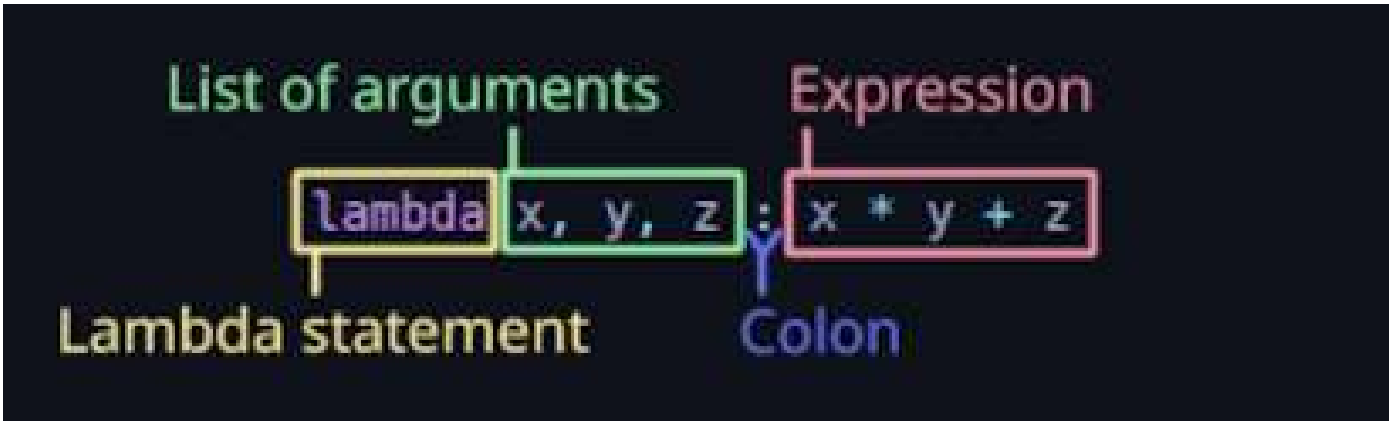
```
[5.770e-01 1.618e+00 2.718e+00 3.140e+00 6.000e+00 3.700e+01 1.729e+03] with the ID number 2668254735376
[5.770e-01 1.618e+00 2.718e+00 3.140e+00 6.000e+00 3.700e+01 1.729e+03] with the ID number 2668254736144
0.577 1.618 2.718 3.14 6.0 37.0 1729.0
0.577 1.618 2.718 3.14 6.0 37.0 1729.0
```

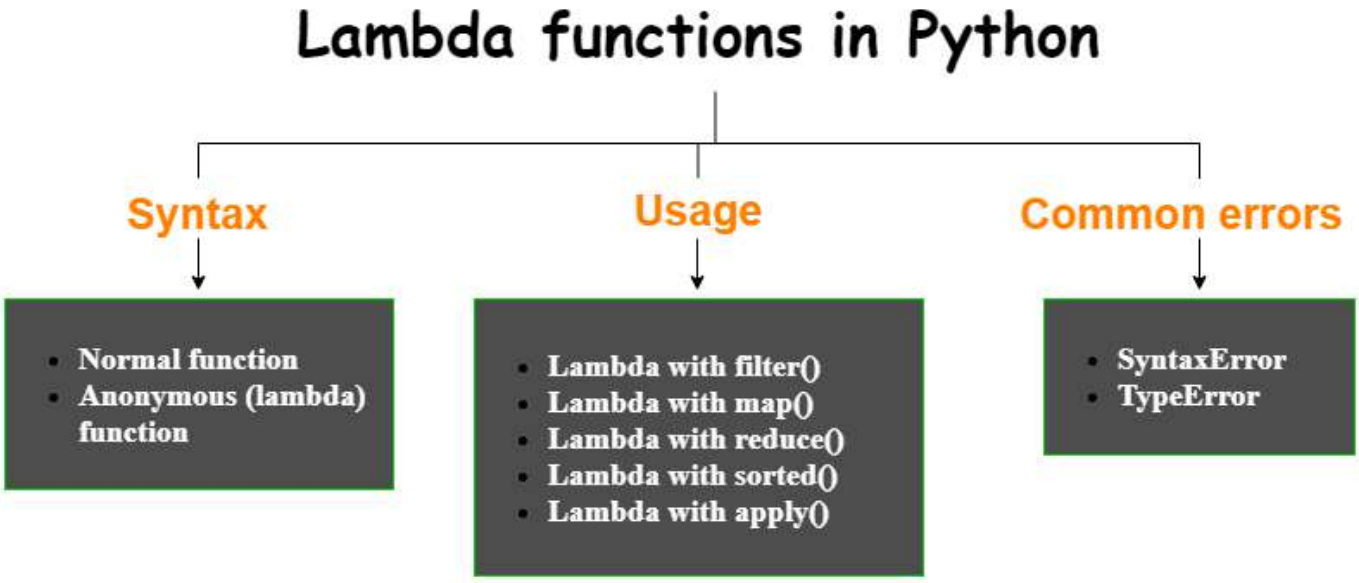
# Python Tutorial

**Created by Mustafa Germec, PhD**

## 17. Lambda Functions in Python

- A **lambda** function is a small anonymous function.
- A **lambda** function can take any number of arguments, but can only have **one expression**.
- The expression is evaluated and returned.
- **Lambda** functions can be used wherever function objects are required.
- Lambda expressions (or lambda functions) are essentially blocks of code that can be assigned to variables, passed as an argument, or returned from a function call, in languages that support high-order functions.
- They have been part of programming languages for quite some time.
- The **main role of the lambda function** is better described in the scenarios when we employ them anonymously inside another function.
- In Python, **the lambda function** can be utilized as an argument to the higher order functions as arguments.



In [1]:

```
# Define a function using 'def'
def f(x):
    return x + 6
print(f(3.14))

# Define the same function using 'lambda'
(lambda x: x+6) (3.14)
```

9.14

Out[1]:

9.14

In [2]:

```
# Define a function using 'def'
def f(x, y):
    return x + y

print('The sum of {} and {} is'.format(3.14, 2.718), f(3.14, 2.718))

# Define the same function using 'lambda
print(f'The sum of pi number and euler number is {(lambda x, y: x+y)(3.14, 2.718)}.')
```

The sum of 3.14 and 2.718 is 5.8580000000000005
The sum of pi number and euler number is 5.8580000000000005.

In [3]:

```
# Calculate the volume of a cube using def and lambda functions
# def function
def cube_volume_def(a):
    return a*a*a

print(f'The volume of a cube using def function is {cube_volume_def(3.14)}.')

# lambda function
print(f'The volume of a cube using lambda function is {(lambda a: a*a*a)(3.14)}.')
```

The volume of a cube using def function is 30.959144000000002.
The volume of a cube using lambda function is 30.959144000000002.

## Multiplication table

In [4]:

```
def mult_table(n):
    return lambda x:x*n

n = int(input('Enter a number: '))
y = mult_table(n)

print(f'The entered number is {n}.')
for i in range(11):
    print(('%d x %d = %d' %(n, i, y(i))))
```

```
Enter a number: 6
The entered number is 6.
6 x 0 = 0
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60
```

## filter()

In [5]:

```
# This program returns a new list when the special numbers in the list are divided by 2 and the remainder is equal to 0
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
list(filter(lambda x:(x%2==0), special_nums))
```

Out[5]:

```
[6, 28]
```

## map()

In [6]:

```
# This program will multiplicate each element of the list with 5 and followed by power of 2.
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(f'Non special numbers are {list(map(lambda x: x*5, special_nums))}')
print(f'Non special numbers list is {list(map(lambda x: pow(x, 2), special_nums))}')
```

```
Non special numbers are [2.885, 8.09, 13.59, 15.700000000000001, 30, 140, 185, 8645]
Non special numbers list is [0.332929, 2.6179240000000004, 7.387524, 9.8596, 36, 784, 1369, 298944
1]
```

## List comprehensions

In [7]:

```
nums = [lambda a=a:a+3.14 for a in range(5)]
nlis = []
for num in nums:
    nlis.append(num())
print(nlis)
```

[3.14, 4.140000000000001, 5.140000000000001, 6.140000000000001, 7.140000000000001]

## lambda function with if/else

In [8]:

```
age = int(input('Enter an age: '))
print(f'The entered age is {age}.')
(lambda age: print('Therefore, you can use a vote.') if (age>=18) else print('Therefore, you do not use a vote.'))(age)
```

Enter an age: 18
The entered age is 18.
Therefore, you can use a vote.

## lambda function usage with multiple statements

In [9]:

```
special_nums = [[0.577, 1.618, 2.718, 3.14], [6, 28, 37, 1729]]
special_nums_sorted = lambda a: (sorted(i) for i in a)

# Get the maximum of special numbers in the list
special_nums_max = lambda a, f: [y[len(y)-1] for y in f(a)]
print(f'The maximum of special numbers in each list is {special_nums_max(special_nums, special_nums_sorted)}.')

# Get the minimum of special numbers in the list
special_nums_min = lambda a, f: [y[len(y)-len(y)] for y in f(a)]
print(f'The minimum of special numbers in each list is {special_nums_min(special_nums, special_nums_sorted)}.')

# Get the second maximum of special numbers in the list
special_nums_second_max = lambda a, f: [y[len(y)-2] for y in f(a)]
print(f'The second maximum of special numbers in each list is {special_nums_second_max(special_nums, special_nur
```

The maximum of special numbers in each list is [3.14, 1729].
The minimum of special numbers in each list is [0.577, 6].
The second maximum of special numbers in each list is [2.718, 37].

## Some examples

In [10]:

```
def func(n):
    return lambda x: x*n

mult_pi_number = func(3.14)
mult_euler_constant = func(0.577)

print(f'The multiplication of euler number and pi number is equal to {mult_pi_number(2.718)}.')
print(f'The multiplication of euler number and euler constant is equal to {mult_euler_constant(2.718)}.')
```

The multiplication of euler number and pi number is equal to 8.53452.
The multiplication of euler number and euler constant is equal to 1.5682859999999998.

In [11]:

```
text = 'Python is a programming language.'
print(lambda text: text)
```

<function <lambda> at 0x00000210C5812820>

In [12]:

```
text = 'Python is a programming language.'
(lambda text: print(text))(text)
```

Python is a programming language.

In [13]:

```
lambda_list = []
# Multiplication of pi number and 12 in one line using lambda function
lambda_list.append((lambda x:x*3.14) (12))
# Division of pi number and 12 in one line using lambda function
lambda_list.append((lambda x: x/3.14) (12))
# Addition of pi number and 12 in one line using lambda function
lambda_list.append((lambda x: x+3.14) (12))
# Subtraction of pi number and 12 in one line using lambda function
lambda_list.append((lambda x: x-3.14) (12))
# Remainder of pi number and 12 in one line using lambda function
lambda_list.append((lambda x: x%3.14) (12))
# Floor division of pi number and 12 in one line using lambda function
lambda_list.append((lambda x: x//3.14) (12))
# Exponential of pi number and 12 in one line using lambda function
lambda_list.append((lambda x: x**3.14) (12))

# Printing the list
print(lambda_list)
```

[37.68, 3.821656050955414, 15.14, 8.86, 2.5799999999999996, 3.0, 2446.972635086879]

In [14]:

```
# Using the function reduce() with lambda to get the sum abd average of the list.
# You should import the library 'functools' first.
import functools
from functools import *
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(f'The sum and average of the numbers in the list are {reduce((lambda a, b: a+b), special_nums)} and {reduce((la
```

The sum and average of the numbers in the list are 1808.0529999999999 and 226.00662499999999, res
pectively.

In [15]:

```
import itertools
from itertools import product
from numpy import sqrt
X=[1]
X1=[2]
Y=[1,2,3]
print(list(product(Y,X,X1)))
print(list(map(lambda x: sqrt(x[1]+x[0]**x[2]),product(Y,X,X1))))
```

[(1, 1, 2), (2, 1, 2), (3, 1, 2)]
[1.4142135623730951, 2.23606797749979, 3.1622776601683795]

In [16]:

```
help(functools)
```

Help on module functools:

NAME
functools - functools.py - Tools for working with functions and callable objects

MODULE REFERENCE
https://docs.python.org/3.9/library/functools (https://docs.python.org/3.9/library/functools)

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.

CLASSES
builtins.object
cached_property
partial
partialmethod

# Math Module Functions in Python

**Created by Mustafa Germec, PhD**

## 18. Math Module Functions in Python

- Python **math module** is defined as the most famous mathematical functions, which includes **trigonometric functions, representation functions, logarithmic functions**, etc.
- Furthermore, it also defines two mathematical constants, i.e., **Pie and Euler number**, etc.
- **Pie (n):** It is a well-known mathematical constant and defined as the ratio of circumstance to the diameter of a circle. Its value is 3.141592653589793.
- **Euler's number(e):**It is defined as the base of the natural logarithmic, and its value is 2.718281828459045.
- The **math** module has a set of methods and constants.

In [1]:

```
# Import math module and functions
import math
from math import *
```

In [2]:

```
# Many functions regarding math modules in python can be find using helf(math) method.
help(math)
```

```
Help on built-in module math:

NAME
  math

DESCRIPTION
  This module provides access to the mathematical functions
  defined by the C standard.

FUNCTIONS
  acos(x, /)
    Return the arc cosine (measured in radians) of x.

    The result is between 0 and pi.

  acosh(x, /)
    Return the inverse hyperbolic cosine of x.

  asin(x, /)
    Return the arc sine (measured in radians) of x.
```

### acos() function

- Return the arc cosine (measured in radians) of x.
- The result is between 0 and pi.
- The parameter must be a double value between -1 and 1.

In [54]:

```
nlis = []
nlis.append(math.acos(1))
nlis.append(math.acos(-1))
print(nlis)
```

[0.0, 3.141592653589793]

## acosh() function

- It is a built-in method defined under the math module to calculate the hyperbolic arc cosine of the given parameter in radians.
- For example, if x is passed as an acosh function (acosh(x)) parameter, it returns the hyperbolic arc cosine value.

In [56]:

```
print(math.acosh(1729))
```

8.148445582615551

## asin() function

- Return the arc sine (measured in radians) of x.
- The result is between -pi/2 and pi/2.

In [52]:

```
nlis = []
nlis.append(math.asin(1))
nlis.append(math.asin(-1))
print(nlis)

```

[1.5707963267948966, -1.5707963267948966]

## asinh() function

- Return the inverse hyperbolic sine of x.

In [55]:

```
print(math.asinh(1729))
```

8.1484457498709

## atan() function

- Return the arc tangent (measured in radians) of x.
- The result is between -pi/2 and pi/2.

In [66]:

```
nlis = []
nlis.append(math.atan(math.inf))      # pozitive infinite
nlis.append(math.atan(-math.inf))     # negative infinite
print(nlis)
```

[1.5707963267948966, -1.5707963267948966]

## atan2() function

- Return the arc tangent (measured in radians) of y/x.
- Unlike atan(y/x), the signs of both x and y are considered.

In [74]:

```
print(math.atan2(1729, 37))
print(math.atan2(1729, -37))
print(math.atan2(-1729, -37))
print(math.atan2(-1729, 37))
print(math.atan2(math.pi, math.inf))
print(math.atan2(math.inf, math.e))
print(math.atan2(math.tau, math.pi))
```

1.5493999395414435
1.5921927140483498
-1.5921927140483498
-1.5493999395414435
0.0
1.5707963267948966
1.1071487177940904

## atanh() function

- Return the inverse hyperbolic tangent of x.

In [91]:

```
nlis=[]
nlis.append(math.atanh(-0.9999))
nlis.append(math.atanh(0))
nlis.append(math.atanh(0.9999))
print(nlis)

```

[-4.951718775643098, 0.0, 4.951718775643098]

## ceil() function

- Rounds a number up to the nearest integer
- Returns the smalles integer greater than or equal to variable.

In [22]:

```
pi_number = math.pi       # math.pi is equal to pi_number 3.14.
print(f'The nearest integer greater than pi number is {math.ceil(pi_number)}.')
```

The nearest integer greater than pi number is 4.

## comb() function

- Number of ways to choose k items from n items without repetition and without order.
- Evaluates to **n!/(k!*(n - k)!)** when **k <= n** and evaluates to zero when **k>n**.
- Also called the binomial coefficient because it is equivalent to the coefficient of k-th term in polynomial expansion of the expression (1 + x)**n.
- Raises **TypeError** if either of the arguments are not integers.
- Raises **ValueError** if either of the arguments are negative.

In [19]:

```
print(f'The combination of 6 with 2 is {math.comb(6, 2)}.')
print(math.comb(10, 3.14))       # It returns a TypeError
```

The combination of 6 with 2 is 15.

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9084/1573976203.py in <module>
      1 print(f'The combination of 6 with 2 is {math.comb(6, 2)}.')
----> 2 print(math.comb(10, 3.14))       # It returns a TypeError

TypeError: 'float' object cannot be interpreted as an integer
```

## copysign() function

- Returns a float consisting of the value of the first parameter and the sign of the second parameter.

In [18]:

```
print(f'The copysign of thes two numbers -3.14 and 2.718 is {math.copysign(-3.14, 2.718)}.')
print(f'The copysign of thes two numbers 1729 and -0.577 is {math.copysign(1729, -0.577)}.')
```

The copysign of thes two numbers -3.14 and 2.718 is 3.14.
The copysign of thes two numbers 1729 and -0.577 is -1729.0.

## cos() function

- Return the cosine of x (measured in radians).

In [105]:

```
print(math.cos(0))
print(math.cos(math.pi/6))
print(math.cos(-1))
print(math.cos(1))
print(math.cos(1729))
print(math.cos(90))
```

```
1.0
0.8660254037844387
0.5403023058681398
0.5403023058681398
0.43204202084333315
-0.4480736161291701
```

## cosh() function

- Return the hyperbolic cosine of x.

In [114]:

```
nlis = []
nlis.append(math.cosh(1))
nlis.append(math.cosh(0))
nlis.append(math.cosh(-5))
print(nlis)
```

```
[1.5430806348152437, 1.0, 74.20994852478785]
```

## degrees() function

- Convert angle x from radians to degrees.

In [122]:

```
nlis = []
nlis.append(math.degrees(math.pi/2))
nlis.append(math.degrees(math.pi))
nlis.append(math.degrees(math.pi/4))
nlis.append(math.degrees(-math.pi))
print(nlis)
```

```
[90.0, 180.0, 45.0, -180.0]
```

## dist() function

- Return the Euclidean distance between two points p and q.
- The points should be specified as sequences (or iterables) of coordinates.
- Both inputs must have the same dimension.
- Roughly equivalent to: **sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))**

In [127]:

```
print(math.dist([30], [60]))
print(math.dist([0.577, 1.618], [3.14, 2.718]))
x = [0.577, 1.618, 2.718]
y = [6, 28, 37]
print(math.dist(x, y))
```

```
30.0
2.7890803143688783
43.59672438383416
```

## erf() function

- Error function at x.
- This method accepts a value between - inf and + inf, and returns a value between - 1 to + 1.

In [136]:

```
nlis = []
nlis.append(math.erf(math.inf))
nlis.append(math.erf(math.pi))
nlis.append(math.erf(math.e))
nlis.append(math.erf(math.tau))
nlis.append(math.erf(0))
nlis.append(math.erf(6))
nlis.append(math.erf(1.618))
nlis.append(math.erf(0.577))
nlis.append(math.erf(-math.inf))
print(nlis)
```

```
[1.0, 0.9999911238536323, 0.9998790689599072, 1.0, 0.0, 1.0, 0.9778739803135315, 0.585500565194
3818, -1.0]
```

## erfc() function

- Complementary error function at x.
- This method accepts a value between - inf and + inf, and returns a value between 0 and 2.

In [137]:

```
nlis = []
nlis.append(math.erfc(math.inf))
nlis.append(math.erfc(math.pi))
nlis.append(math.erfc(math.e))
nlis.append(math.erfc(math.tau))
nlis.append(math.erfc(0))
nlis.append(math.erfc(6))
nlis.append(math.erfc(1.618))
nlis.append(math.erfc(0.577))
nlis.append(math.erfc(-math.inf))
print(nlis)
```

```
[0.0, 8.876146367641612e-06, 0.00012093104009276267, 6.348191705159502e-19, 1.0, 2.1519736712
498913e-17, 0.022126019686468514, 0.41449943480561824, 2.0]
```

## exp() function

- The **math.exp()** method returns **E** raised to the power of x (Ex).
- **E** is the base of the natural system of logarithms (approximately 2.718282) and x is the number passed to it.

In [139]:

```
nlis = []
nlis.append(math.exp(math.inf))
nlis.append(math.exp(math.pi))
nlis.append(math.exp(math.e))
nlis.append(math.exp(math.tau))
nlis.append(math.exp(0))
nlis.append(math.exp(6))
nlis.append(math.exp(1.618))
nlis.append(math.exp(0.577))
nlis.append(math.exp(-math.inf))
print(nlis)
```

```
[inf, 23.140692632779267, 15.154262241479262, 535.4916555247646, 1.0, 403.4287934927351, 5.042
994235377287, 1.780688344599613, 0.0]
```

## expm1() function

- Return exp(x)-1.
- This function avoids the loss of precision involved in the direct evaluation of exp(x)-1 for small x.

In [141]:

```
nlis = []
nlis.append(math.expm1(math.inf))
nlis.append(math.expm1(math.pi))
nlis.append(math.expm1(math.e))
nlis.append(math.expm1(math.tau))
nlis.append(math.expm1(0))
nlis.append(math.expm1(6))
nlis.append(math.expm1(1.618))
nlis.append(math.expm1(0.577))
nlis.append(math.expm1(-math.inf))
print(nlis)
```

[inf, 22.140692632779267, 14.154262241479262, 534.4916555247646, 0.0, 402.4287934927351, 4.042
994235377287, 0.7806883445996128, 6.38905609893065, -1.0]

## fabs() function

- Returns the absolute value of a number

In [14]:

```
print(f'The absolute value of the number -1.618 is {math.fabs(-1.618)}.')
```

The absolute value of the number -1.618 is 1.618.

## factorial() function

- Returns the factorial of a number.

In [28]:

```
print(f'The factorial of the number 6 is {math.factorial(6)}.')
```

The factorial of the number 6 is 720.

In [29]:

```
# Factorial of negative numbers returns a ValueError.
print(math.factorial(-6))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9084/3052214312.py in <module>
      1 # Factorial of negative numbers returns a ValueError.
----> 2 print(math.factorial(-6))

ValueError: factorial() not defined for negative values
```

In [30]:

```
# Factorial of non-unteger numbers returns a TypeError.
print(math.factorial(3.14))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9084/3264451390.py in <module>
      1 # Factorial of non-unteger numbers returns a TypeError.
----> 2 print(math.factorial(3.14))

TypeError: 'float' object cannot be interpreted as an integer
```

## floor() functions:

- Rounds a number down to the nearest integer

In [34]:

```
print(math.floor(3.14))
```

```
3
```

## fmod() function

- Returns the remainder of x/y

In [37]:

```
print(math.fmod(37, 6))
print(math.fmod(1728, 37))
```

```
1.0
26.0
```

## frexp() function

- Returns the mantissa and the exponent, of a specified number

In [31]:

```
print(math.frexp(2.718))
```

```
(0.6795, 2)
```

## fsum() function

- Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)

In [142]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(math.fsum(special_nums))
```

1808.053

## gamma() function

- Returns the gamma function at x.
- You can find more information about **gamma function** from this Link. (https://en.wikipedia.org/wiki/Gamma_function)

In [143]:

```
print(math.gamma(3.14))
print(math.gamma(6))
print(math.gamma(2.718))
```

2.2844806338178008
120.0
1.5671127417668826

## gcd() function

- Returns the greatest common divisor of two integers

In [144]:

```
print(math.gcd(3, 10))
print(math.gcd(4, 8))
print(math.gcd(0, 0))
```

1
4
0

## hypot() function

- Returns the Euclidean norm.
- Multidimensional Euclidean distance from the origin to a point.
- Roughly equivalent to: sqrt(sum(x**2 for x in coordinates))
- For a two dimensional point (x, y), gives the hypotenuse using the Pythagorean theorem: sqrt(x*x* + *yy*).

In [148]:

```
print(math.hypot(3, 4))
print(math.hypot(5, 12))
print(math.hypot(8, 15))
```

```
5.0
13.0
17.0
```

## isclose() function

- It checks whether two values are close to each other, or not.
- Returns True if the values are close, otherwise False.
- This method uses a relative or absolute tolerance, to see if the values are close.
- **Tip:** It uses the following formula to compare the values: abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)

In [11]:

```
print(math.isclose(math.pi, math.tau))     # tau number is 2 times higher than pi number
print(math.isclose(3.14, 2.718))
print(math.isclose(3.14, 1.618))
print(math.isclose(10, 5, rel_tol = 3, abs_tol=0))
print(math.isclose(3.14, 3.1400000000001))
```

```
False
False
False
True
True
```

## isfinite() function

- Return *True* if x is neither an **infinity** nor a **NaN**, and *False* otherwise.

In [155]:

```
nlis = []
nlis.append(math.isfinite(math.inf))
nlis.append(math.isfinite(math.pi))
nlis.append(math.isfinite(math.e))
nlis.append(math.isfinite(math.tau))
nlis.append(math.isfinite(0))
nlis.append(math.isfinite(6))
nlis.append(math.isfinite(1.618))
nlis.append(math.isfinite(0.577))
nlis.append(math.isfinite(-math.inf))
nlis.append(math.isfinite(float('NaN')))
nlis.append(math.isfinite(float('inf')))
print(nlis)
```

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

## isinf() function

- Return *True* if x is a **positive or negative infinity**, and *False* otherwise.

In [161]:

```
nlis = []
nlis.append(math.isinf(math.inf))
nlis.append(math.isinf(math.pi))
nlis.append(math.isinf(math.e))
nlis.append(math.isinf(math.tau))
nlis.append(math.isinf(0))
nlis.append(math.isinf(6))
nlis.append(math.isinf(1.618))
nlis.append(math.isinf(0.577))
nlis.append(math.isinf(-math.inf))
print(nlis)
```

[True, False, False, False, False, False, False, False, True]

## isnan() function

- Return *True* if x is a **NaN (not a number)**, and *False* otherwise.

In [162]:

```
nlis = []
nlis.append(math.isnan(float('NaN')))
nlis.append(math.isnan(math.inf))
nlis.append(math.isnan(math.pi))
nlis.append(math.isnan(math.e))
nlis.append(math.isnan(math.tau))
nlis.append(math.isnan(0))
nlis.append(math.isnan(6))
nlis.append(math.isnan(1.618))
nlis.append(math.isnan(0.577))
nlis.append(math.isnan(-math.inf))
nlis.append(math.isnan(math.nan))
print(nlis)
```

[True, False, False, False, False, False, False, False, False, False, True]

## isqrt() function

- Rounds a square root number downwards to the nearest integer.
- The returned square root value is the floor value of square root of a non-negative integer number.
- It gives a **ValueError** and **TypeError** when **a negative integer number** and **a float number** are used, respectively.

In [15]:

```
print(math.isqrt(4))
print(math.isqrt(5))
print(math.isqrt(-5))
```

```
2
2

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20068/2393595883.py in <module>
      1 print(math.isqrt(4))
      2 print(math.isqrt(5))
----> 3 print(math.isqrt(-5))

ValueError: isqrt() argument must be nonnegative
```

In [16]:

```
print(math.isqrt(3.14))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20068/4116779010.py in <module>
----> 1 print(math.isqrt(3.14))

TypeError: 'float' object cannot be interpreted as an integer
```

## lcm() function

- Least Common Multiple.

In [168]:

```
nlis = []
nlis.append(math.lcm(3, 5, 25))
nlis.append(math.lcm(9, 6, 27))
nlis.append(math.lcm(21, 27, 54))
print(nlis)
```

```
[75, 54, 378]
```

## ldexp() function

- Returns the inverse of **math.frexp()** which is x*(2^i) of the given numbers x and i

In [19]:

```
print(math.ldexp(20, 4))
print(20*(2**4))
```

```
320.0
320
```

## lgamma() function

- Returns the log gamma value of x

In [26]:

```
print(math.gamma(6))
print(math.lgamma(6))
print(math.log(120))      # print(math.gamma(6)) = 120
```

```
120.0
4.787491742782047
4.787491742782046
```

## log() function

- log(x, [base=math.e])
- Return the logarithm of x to the given base.

In [174]:

```
nlis = []
nlis.append(math.log(90))
nlis.append(math.log(1))
nlis.append(math.log(math.e))
nlis.append(math.log(math.pi))
nlis.append(math.log(math.tau))
nlis.append(math.log(math.inf))
nlis.append(math.log(math.nan))
print(nlis)
```

```
[4.499809670330265, 0.0, 1.0, 1.1447298858494002, 1.8378770664093453, inf, nan]
```

## log10() function

- Return the base 10 logarithm of x.

In [177]:

```
nlis = []
nlis.append(math.log10(90))
nlis.append(math.log10(1))
nlis.append(math.log10(math.e))
nlis.append(math.log10(math.pi))
nlis.append(math.log10(math.tau))
nlis.append(math.log10(math.inf))
nlis.append(math.log10(math.nan))
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]:

```
nlis = []
nlis.append(math.log1p(90))
nlis.append(math.log1p(1))
nlis.append(math.log1p(math.e))
nlis.append(math.log1p(math.pi))
nlis.append(math.log1p(math.tau))
nlis.append(math.log1p(math.inf))
nlis.append(math.log1p(math.nan))
print(nlis)
```

[4.51085950651685, 0.6931471805599453, 1.3132616875182228, 1.4210804127942926, 1.985568308
7099187, inf, nan]

## log2() function

- Return the base 2 logarithm of x.

In [183]:

```
nlis = []
nlis.append(math.log2(90))
nlis.append(math.log2(2))
nlis.append(math.log2(1))
nlis.append(math.log2(math.e))
nlis.append(math.log2(math.pi))
nlis.append(math.log2(math.tau))
nlis.append(math.log2(math.inf))
nlis.append(math.log2(math.nan))
print(nlis)
```

[6.491853096329675, 1.0, 0.0, 1.4426950408889634, 1.6514961294723187, 2.651496129472319, inf, n
an]

## modf() function

- It returns the frwactional and integer parts of the certain number. Both the outputs carry the sign of x and are of type float.

In [29]:

```
print(math.modf(math.pi))
print(math.modf(math.e))
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]:

```
nlis = []
nlis.append(math.nextafter(3.14, 90))
nlis.append(math.nextafter(6, 2.718))
nlis.append(math.nextafter(3, math.e))
nlis.append(math.nextafter(28, math.inf))
nlis.append(math.nextafter(1.618, math.nan))
nlis.append(math.nextafter(1, 1))
nlis.append(math.nextafter(0, 0))
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]:

```
print(math.perm(6, 2))
print(math.perm(6, 6))
```

30
720

## pow() function

- Returns the value of x to the power of y.

In [34]:

```
print(math.pow(10, 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]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 28, 37, 1729]
print(math.prod(special_nums))
```

```
85632659.07026622
```

## radians() function

- Convert angle x from degrees to radians.

In [193]:

```
nlis = []
nlis.append(math.radians(0))
nlis.append(math.radians(30))
nlis.append(math.radians(45))
nlis.append(math.radians(60))
nlis.append(math.radians(90))
nlis.append(math.radians(120))
nlis.append(math.radians(180))
nlis.append(math.radians(270))
nlis.append(math.radians(360))
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 - n*y where n*y 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]:

```
nlis = []
nlis.append(math.remainder(3.14, 2.718))
nlis.append(math.remainder(6, 28))
nlis.append(math.remainder(5, 3))
nlis.append(math.remainder(1729, 37))
print(nlis)
```

[0.42200000000000015, 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]:

```
nlis = []
nlis.append(math.sin(math.pi))
nlis.append(math.sin(math.pi/2))
nlis.append(math.sin(math.e))
nlis.append(math.sin(math.nan))
nlis.append(math.sin(math.tau))
nlis.append(math.sin(30))
nlis.append(math.sin(-5))
nlis.append(math.sin(37))
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]:

```
nlis = []
nlis.append(math.sinh(1))
nlis.append(math.sinh(0))
nlis.append(math.sinh(-5))
nlis.append(math.sinh(math.pi))
nlis.append(math.sinh(math.e))
nlis.append(math.sinh(math.tau))
nlis.append(math.sinh(math.nan))
nlis.append(math.sinh(math.inf))
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]:

```
nlis = []
nlis.append(math.sqrt(1))
nlis.append(math.sqrt(0))
nlis.append(math.sqrt(37))
nlis.append(math.sqrt(math.pi))
nlis.append(math.sqrt(math.e))
nlis.append(math.sqrt(math.tau))
nlis.append(math.sqrt(math.nan))
nlis.append(math.sqrt(math.inf))
print(nlis)
```

```
[1.0, 0.0, 6.082762530298219, 1.7724538509055159, 1.6487212707001282, 2.5066282746310002, na
n, inf]
```

## tan() function

- Return the tangent of x (measured in radians).

In [212]:

```
nlis = []
nlis.append(math.tan(0))
nlis.append(math.tan(30))
nlis.append(math.tan(45))
nlis.append(math.tan(60))
nlis.append(math.tan(90))
nlis.append(math.tan(120))
nlis.append(math.tan(180))
nlis.append(math.tan(270))
nlis.append(math.tan(360))
print(nlis)
```

```
[0.0, -6.405331196646276, 1.6197751905438615, 0.320040389379563, -1.995200412208242, 0.71312
30097859091, 1.3386902103511544, -0.17883906379845224, -3.380140413960958]
```

## tanh() function

- Return the hyperbolic tangent of x.

In [214]:

```
nlis = []
nlis.append(math.tanh(1))
nlis.append(math.tanh(0))
nlis.append(math.tanh(-5))
nlis.append(math.tanh(math.pi))
nlis.append(math.tanh(math.e))
nlis.append(math.tanh(math.tau))
nlis.append(math.tanh(math.nan))
nlis.append(math.tanh(math.inf))
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]:

```
nlis = []
nlis.append(math.trunc(1))
nlis.append(math.trunc(0))
nlis.append(math.trunc(-5))
nlis.append(math.trunc(0.577))
nlis.append(math.trunc(1.618))
nlis.append(math.trunc(math.pi))
nlis.append(math.trunc(math.e))
nlis.append(math.trunc(math.tau))
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]:

```
import sys
nlis = []
nlis.append(math.ulp(1))
nlis.append(math.ulp(0))
nlis.append(math.ulp(-5))
nlis.append(math.ulp(0.577))
nlis.append(math.ulp(1.618))
nlis.append(math.ulp(math.pi))
nlis.append(math.ulp(math.e))
nlis.append(math.ulp(math.tau))
nlis.append(math.ulp(math.nan))
nlis.append(math.ulp(math.inf))
nlis.append(math.ulp(-math.inf))
nlis.append(math.ulp(float('nan')))
nlis.append(math.ulp(float('inf')))
x = sys.float_info.max
nlis.append(math.ulp(x))
print(nlis)
```

```
[2.220446049250313e-16, 5e-324, 8.881784197001252e-16, 1.1102230246251565e-16, 2.2204460492
50313e-16, 4.440892098500626e-16, 4.440892098500626e-16, 8.881784197001252e-16, nan, inf, inf, n
an, 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]:

```
import math
from math import *
```

In [24]:

```
# Using for loop
cubic_nums = []
for i in range(5):
    i**=3
    cubic_nums.append(i)
print(cubic_nums)

# Using list comprehension
cubic_nums = [i**3 for i in range(5)]
print(cubic_nums)

# Using list comprehension
cubic_nums = [math.pow(i, 3) for i in range(5)]
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]:

```
# Using for loop
even_numbers = []
for i in range(21):
    if i%2 == 0:
        even_numbers.append(i)
print(even_numbers)

# Using list comprehension
even_numbers = [i for i in range(21) if i%2==0]
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 insexts 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 [31]:

```
# Using for loop
n = int(input('Enter a number: '))
print(f'The entered number is {n}.')
insect_nums = []
for i in range(12):
    i = n*(2**i)
    insect_nums.append(i)
print(insect_nums)

# Using list comprehension
insect_nums = [n*(2**i) for i in range(0, 12)]
print(insect_nums)
```

```
The entered number is 10.
[10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480]
[10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480]
```

In [20]:

```
# Create a list of multiplies of three from 0 to 30
# Using for loop
nlis =[]
for i in range(30):
    if i%3 == 0:
        nlis.append(i)
print(nlis)

# Using list comprehension
nlis = [i for i in range(30) if i%3==0]
print(nlis)
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]
```

In [19]:

```
# Using for loop
text = []
for i in 'Python is a programming language':
    text.append(i)
print(text)

# Using list comprehension
text = [i for i in 'Python is a programming language']
print(text)
```

```
['P', 'y', 't', 'h', 'o', 'n', ' ', 'i', 's', ' ', 'a', ' ', 'p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', ' ', 'l', 'a', 'n', 'g', 'u', 'a', 'g',
'e']
['P', 'y', 't', 'h', 'o', 'n', ' ', 'i', 's', ' ', 'a', ' ', 'p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g', ' ', 'l', 'a', 'n', 'g', 'u', 'a', 'g',
'e']
```

In [33]:

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

#Using for loop
three_times = []
for i in special_nums:
    i*=3
    three_times.append(i)
print(three_times)

# Using list comprehension
three_times = [x*3 for x in special_nums]
print(three_times)
```

```
[1.7309999999999999, 4.854, 8.154, 9.42, 18, 84, 111, 5187]
[1.7309999999999999, 4.854, 8.154, 9.42, 18, 84, 111, 5187]
```

In [39]:

```
# Using for loop
languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']
lang_lis = []
for i in languages:
    if 't' in i:
        lang_lis.append(i)
print(lang_lis)

#Using list comprehension
lang_lis = [i for i in languages if 't' in i]
print(lang_lis)
```

```
['Python', 'JavaScript']
['Python', 'JavaScript']
```

In [46]:

```
languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']
# Using for loop
lang_lis = []
for i in languages:
    if i != 'C':
        lang_lis.append(i)
print(lang_lis)

#Using list comprehension
lang_lis = [i for i in languages if i != 'C']
print(lang_lis)
```

```
['Python', 'Java', 'JavaScript', 'C++', 'PHP']
['Python', 'Java', 'JavaScript', 'C++', 'PHP']
```

In [48]:

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

# Using for loop
new_lis = []
for i in special_nums:
    if i < 5:
        new_lis.append(i)
print(new_lis)

# Using list comprehension
new_lis = [i for i in special_nums if i < 5]
print(new_lis)
```

```
[0.577, 1.618, 2.718, 3.14]
[0.577, 1.618, 2.718, 3.14]
```

In [51]:

```
languages = ['Java', 'JavaScript', 'C', 'C++', 'PHP']
# Using for loop
lang_lis = []
for i in languages:
    i = 'Python'
    lang_lis.append(i)
print(lang_lis)

#Using list comprehension
lang_lis = ['Python' for i in languages]
print(lang_lis)
```

```
['Python', 'Python', 'Python', 'Python', 'Python']
['Python', 'Python', 'Python', 'Python', 'Python']
```

In [53]:

```
languages = ['Java', 'JavaScript', 'C', 'C++', 'PHP']
# Using for loop
lang_lis = []
for i in languages:
    if i != 'Java':
        lang_lis.append(i)
    else:
        lang_lis.append('Python')
print(lang_lis)


#Using list comprehension
lang_lis = [i if i != 'Java' else 'Python' for i in languages]
print(lang_lis)
```

```
['Python', 'JavaScript', 'C', 'C++', 'PHP']
['Python', 'JavaScript', 'C', 'C++', 'PHP']
```

In [56]:

```
languages = ['Python', 'Java', 'JavaScript', 'C', 'C++', 'PHP']
# Using for loop
lang_lis = []
for i in languages:
    i = i.upper()
    lang_lis.append(i)
print(lang_lis)

#Using list comprehension
lang_lis = [i.upper() for i in languages]
print(lang_lis)
```

```
['PYTHON', 'JAVA', 'JAVASCRIPT', 'C', 'C++', 'PHP']
['PYTHON', 'JAVA', 'JAVASCRIPT', 'C', 'C++', 'PHP']
```

In [59]:

```
# Using for loop
python = []
for i in 'Python':
    python.append(i)
print(python)

# Using list comprehension
python = [i for i in 'Python']
print(python)

# Using lambda function
python = list(map(lambda i: i, 'Python'))
print(python)
```

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

In [85]:

```
# Using for loop
numbers = []
for i in range(11):
    if i%2 == 0:
        numbers.append('Even')
    else:
        numbers.append('Odd')
print(f'For loop: {numbers}')

# Using list comprehension
numbers = ['Even' if i%2==0  else 'Odd' for i in range(11)]
print(f'List comprehension: {numbers}')

# Using lambda function
numbers = list(map(lambda i: i, ['Even' if i%2==0  else 'Odd' for i in range(11)]))
print(f'Lambda: {numbers}')
```

```
For loop: ['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even']
List comprehension: ['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even']
Lambda: ['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even']
```

In [79]:

```
# Using nested for loop
empty_list = []
matrix_list = [[0.577, 1.618, 2.718, 3.14], [6, 28, 37, 1729]]

for i in range(len(matrix_list[0])):
    T_row = []
    for row in matrix_list:
        T_row.append(row[i])
    empty_list.append(T_row)
print(empty_list)

# Using list comprehension
empty_list = [[row[i] for row in matrix_list] for i in range(4)]
print(empty_list)
```

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

In [82]:

```
# Using nested for loop
empty_matrix = []
for i in range(5):
    empty_matrix.append([])
    for j in range(5):
        empty_matrix[i].append(j)
print(empty_matrix)

# Using list comprehension
empty_matrix = [[j for j in range(5)] for i in range(5)]
print(empty_matrix)
```

```
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
[[0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4], [0, 1, 2, 3, 4]]
```

In [89]:

```
# Transpose of 2D matrix
matrix = [[0.577, 1.618, 0],
          [2.718, 3.14, 1],
          [6, 28, 28]]
transpose_matrix = [[i[j] for i in matrix] for j in range(len(matrix))]
print(transpose_matrix)
```

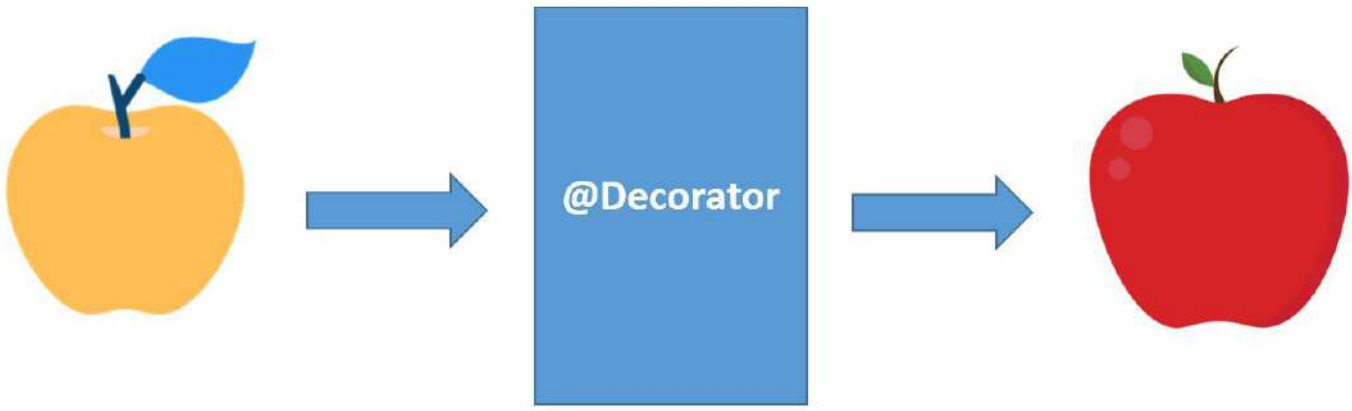
```
[[0.577, 2.718, 6], [1.618, 3.14, 28], [0, 1, 28]]
```

# 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 [ ]:

```
"""
@hello_decorator
def hi_decorator():
    print("Hello")
"""

'''
Above code is equal to -

def hi_decorator():
    print("Hello")

hi_decorator = hello_decorator(hi_decorator)
'''
```

In [5]:

```
# Import libraries
import decorator
from decorator import *
import functools
import math
```

In [26]:

```
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]:

```
# Define a function
"""
In the following function, when the code was executed, it yeilds the outputs for both functions.
The function new_text() alluded to the function mytext() and behave as function.
"""
def mytext(text):
    print(text)

mytext('Python is a programming language.')
new_text = mytext
new_text('Hell, Python!')
```

```
Python is a programming language.
Hell, Python!
```

In [1]:

```
def multiplication(num):
    return num * num

mult = multiplication
mult(3.14)
```

Out[1]:

9.8596

## Nested/Inner Function

In [28]:

```
# Define a function
"""
In the following function, it is nonsignificant how the child functions are announced.
The implementation of the child function does influence on the output.
These child functions are topically linked with the function mytext(), therefore they can not be called individually.
"""
def mytext():
    print('Python is a programming language.')
    def new_text():
        print('Hello, Python!')
    def message():
        print('Hi, World!')

    new_text()
    message()
mytext()

```

Python is a programming language.
Hello, Python!
Hi, World!

In [3]:

```
# Define a function
"""
In the following example, the function text() is nesred into the function message().
It will return each time when the function tex() is called.
"""
def message():
    def text():
        print('Python is a programming language.')
    return text

new_message = message()
new_message()
```

Python is a programming language.

In [4]:

```
def function(num):
    def mult(num):
        return num*num

    output = mult(num)
    return output
mult(3.14)
```

Out[4]:

9.8596

In [13]:

```
def msg(text):
    'Hello, World!'
    def mail():
        'Hi, Python!'
        print(text)

    mail()

msg('Python is the most popular programming language.')
```

Python is the most popular programming language.

## Passing functions

In [29]:

```
# Define a function
"""
In this function, the mult() and divide() functions as argument in operator() function are passed.
"""
def mult(x):
    return x * 3.14
def divide(x):
    return x/3.14
def operator(function, x):
    number = function(x)
    return number

print(operator(mult, 2.718))
print(operator(divide, 1.618))
```

8.53452
0.5152866242038217

In [7]:

```
def addition(num):
    return num + math.pi

def called_function(func):
    added_number = math.e
    return func(added_number)

called_function(addition)
```

Out[7]:

```
5.859874482048838
```

In [111]:

```
def decorator_one(function):
    def inner():
        num = function()
        return num * (num**num)
    return inner

def decorator_two(function):
    def inner():
        num = function()
        return (num**num)/num
    return inner

@decorator_one
@decorator_two
def number():
    return 4

print(number())

 # The above decorator returns the following code
x = pow(4, 4)/4
print(x*(x**x))
```

```
2.5217283965692467e+117
2.5217283965692467e+117
```

## Functions reverting other functions

In [11]:

```
def msg_func():
    def text():
        return "Python is a programming language."
    return text
msg = msg_func()
print(msg())
```

```
Python is a programming language.
```

## Decorating functions

In [8]:

```
# Define a decorating function
"""
In the following example, the function outer_addition that is some voluminous is decorated.
"""
def addition(a, b):
    print(a+b)
def outer_addition(func):
    def inner(a, b):
        if a < b:
            a, b = b, a
        return func(a, b)
    return inner

result = outer_addition(addition)
result(math.pi, math.e)
```

5.859874482048838

In [9]:

```
"""
Rather than above function, Python ensures to employ decorator in easy way with the symbol @ called 'pie' syntax, as
"""
def outer_addition(function):
    def inner(a, b):
        if a < b:
            a, b = b, a
        return function(a, b)
    return inner

@outer_addition        # Syntax of decorator
def addition(a, b):
    print(a+b)
result = outer_addition(addition)
result(math.pi, math.e)
```

5.859874482048838

In [17]:

```
def decorator_text_uppercase(func):
    def wrapper():
        function = func()
        text_uppercase = function.upper()
        return text_uppercase

    return wrapper

# Using a function
def text():
    return 'Python is the most popular programming language.'

decorated_result = decorator_text_uppercase(text)
print(decorated_result())

# Using a decorator
@decorator_text_uppercase
def text():
    return 'Python is the most popular programming language.'

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]:

```
def do_twice(function):
    def wrapper_do_twice():
        function()
        function()
    return wrapper_do_twice

@do_twice
def text():
    print('Python is a programming language.')
text()
```

```
Python is a programming language.
Python is a programming language.
```

## Decorators with Arguments

In [39]:

```
def do_twice(function):
    """
    The function wrapper_function() can admit any number of argument and pass them on the function.
    """
    def wrapper_function(*args, **kwargs):
        function(*args, **kwargs)
        function(*args, **kwargs)
    return wrapper_function

@do_twice
def text(programming_language):
    print(f'{programming_language} is a programming language.')
text('Python')
```

```
Python is a programming language.
Python is a programming language.
```

## Returning values from decorated function

In [41]:

```
@do_twice
def returning(programming_language):
    print('Python is a programming language.')
    return f'Hello, {programming_language}'

hello_python = returning('Python')
```

```
Python is a programming language.
Python is a programming language.
```

## Fancy decorators

- @propertymethod
- @staticmethod
- @classmethod

In [45]:

```
class Microorganism:
    def __init__(self, name, product):
        self.name = name
        self.product = product
    @property
    def show(self):
        return self.name + ' produces ' + self.product + ' enzyme'

organism = Microorganism('Aspergillus niger', 'inulinase')
print(f'Microorganism name: {organism.name}')
print(f'Microorganism product: {organism.product}')
print(f'Message: {organism.show}.')
```

```
Microorganism name: Aspergillus niger
Microorganism product: inulinase
Message: Aspergillus niger produces inulinase enzyme.
```

In [46]:

```
class Micoorganism:
    @staticmethod
    def name():
        print('Aspergillus niger is a fungus that produces inulinase enzyme.')

organims = Micoorganism()
organims.name()
Micoorganism.name()
```

```
Aspergillus niger is a fungus that produces inulinase enzyme.
Aspergillus niger is a fungus that produces inulinase enzyme.
```

In [97]:

```
class Microorganism:
    def __init__(self, name, product):
        self.name = name
        self.product = product

    @classmethod
    def display(cls):
        return cls('Aspergillus niger', 'inulinase')

organism = Microorganism.display()
print(f'The fungus {organism.name} produces {organism.product} enzyme.')

```

```
The fungus Aspergillus niger produces inulinase enzyme.
```

## Decorator with arguments

In [49]:

```
"""
In the following example, @iterate refers to a function object that can be called in another function.
The @iterate(numbers=4) will return a function which behaves as a decorator.
"""
def iterate(numbers):
    def decorator_iterate(function):
        @functools.wraps(function)
        def wrapper(*args, **kwargs):
            for _ in range(numbers):
                worth = function(*args, **kwargs)
            return worth
        return wrapper
    return decorator_iterate

@iterate(numbers=4)
def function_one(name):
    print(f'{name}')

x = function_one('Python')
```

```
Python
Python
Python
Python
```

In [21]:

```
def arguments(func):
    def wrapper_arguments(argument_1, argument_2):
        print(f'The arguments are {argument_1} and {argument_2}.')
        func(argument_1, argument_2)
    return wrapper_arguments


@arguments
def programing_language(lang_1, lang_2):
    print(f'My favorite programming languages are {lang_1} and {lang_2}.')

programing_language("Python", "R")
```

```
The arguments are Python and R.
My favorite programming languages are Python and R.
```

## Multiple decorators

In [18]:

```
def splitted_text(text):
    def wrapper():
        function = text()
        text_splitting = function.split()
        return text_splitting

    return wrapper

@splitted_text
@decorator_text_uppercase      # Calling other decorator above
def text():
    return 'Python is the most popular programming language.'
text()
```

Out[18]:

['PYTHON', 'IS', 'THE', 'MOST', 'POPULAR', 'PROGRAMMING', 'LANGUAGE.']

## Arbitrary arguments

In [43]:

```
def arbitrary_argument(func):
    def wrapper(*args,**kwargs):
        print(f'These are positional arguments {args}.')
        print(f'These are keyword arguments {kwargs}.')
        func(*args)
    return wrapper

"""1. Without arguments decorator"""
print(__doc__)
@arbitrary_argument
def without_argument():
    print("There is no argument in this decorator.")

without_argument()

"""2. With positional arguments decorator"""
print(__doc__)
@arbitrary_argument
def with_positional_argument(x1, x2, x3, x4, x5, x6):
    print(x1, x2, x3, x4, x5, x6)

with_positional_argument(math.inf, math.tau, math.pi, math.e, math.nan, -math.inf)

"""3. With keyword arguments decorator"""
print(__doc__)
@arbitrary_argument
def with_keyword_argument():
    print("Python and R are my favorite programming languages and keyword arguments.")

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, na
n, -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]:

```
def capitalize_dec(function):
    @functools.wraps(function)
    def wrapper():
        return function().capitalize()
    return wrapper

@capitalize_dec
def message():
    "Python is the most popular programming language."
    return 'PYTHON IS THE MOST POPULAR PROGRAMMING LANGUAGE. '

print(message())
print()
print(message.__name__)
print(message.__doc__)
```

Python is the most popular programming language.

message
Python is the most popular programming language.

## Preserving decorators

In [85]:

```
def preserved_decorator(function):
    def wrapper():
        print('Before calling the function, this is printed.')
        function()
        print('After calling the function, this is printed.')
    return wrapper

@preserved_decorator
def message():
    """This function prints the message when it is called."""
    print('Python is the most popular programming language.')

message()
print(message.__name__)
print(message.__doc__)
print(message.__class__)
print(message.__module__)
print(message.__code__)
print(message.__closure__)
print(message.__annotations__)
print(message.__dir__)
print(message.__format__)
```

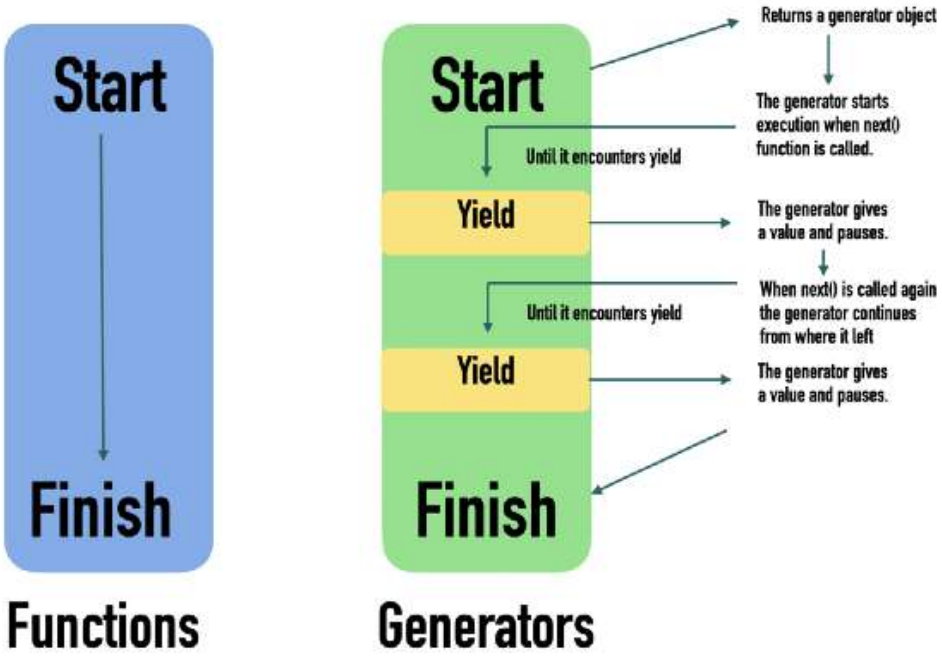
```
Before calling the function, this is printed.
Python is the most popular programming language.
After calling the function, this is printed.
wrapper
None
<class 'function'>
__main__
<code object wrapper at 0x0000029986F96970, file "C:\Users\test\AppData\Local\Temp/ipykernel_118
0/3788198392.py", line 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 tha traversal object and a simple way of creating iterators.
- It traverses the entire items at once.
- The generator can alos be an expression in which syntax is smilar 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 lenghty process to create iterators.
- That is why the generator plays a significant role in simplfying 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 responsibel 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]:

```
def function():
    for i in range(10):
        if i%2==0:
            yield i

nlis = []
for i in function():
    nlis.append(i)
print(nlis)
```

[0, 2, 4, 6, 8]

In [26]:

```
def func():
    for i in range(25):
        if i%4==0:
            yield i

num_lis = []
for i in func():
    num_lis.append(i)
print(num_lis)
```

[0, 4, 8, 12, 16, 20, 24]

In [2]:

```
def message():
    msg_one = 'Hello, World!'
    yield msg_one

    msg_two = 'Hi, Python!'
    yield msg_two

    msg_three = 'Python is the most popular programming language.'
    yield msg_three

result = message()
print(next(result))
print(next(result))
print(next(result))
```

Hello, World!
Hi, Python!
Python is the most popular programming language.

In [4]:

```
"""
In the following example, the list comprehension will return the list of cube of elements.
Whereas the generator expression will return the reference of the calculated value.
Rather than this application, the ^function 'next()' can be used on the generator object.
"""
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 37, 1729]

list_comp = [i*3 for i in special_nums]     # This is a list comprehension.
generator_exp = (i*3 for i in special_nums) # This is a generator expression.

print(list_comp)
print(generator_exp)
```

```
[1.7309999999999999, 4.854, 8.154, 9.42, 18, 111, 5187]
<generator object <genexpr> at 0x000002572F0E1230>
```

In [8]:

```
special_nums = [0.577, 1.618, 2.718, 3.14, 6, 37, 1729]

generator_exp = (i*3 for i in special_nums) # This is a generator expression.

nums_list = []
nums_list.append(next(generator_exp))
nums_list.append(next(generator_exp))
nums_list.append(next(generator_exp))
nums_list.append(next(generator_exp))
nums_list.append(next(generator_exp))
nums_list.append(next(generator_exp))
nums_list.append(next(generator_exp))
print(nums_list)
```

```
[1.7309999999999999, 4.854, 8.154, 9.42, 18, 111, 5187]
```

In [12]:

```
def mult_table(n):
    for i in range(0, 11):
        yield n*i
        i+=1

mult_table_list = []
for i in mult_table(20):
    mult_table_list.append(i)
print(mult_table_list)
```

```
[0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200]
```

In [17]:

```
import sys

# List comprehension
cubic_nums_lc = [i**3 for i in range(1500)]
print(f'Memory in bytes with list comprehension is {sys.getsizeof(cubic_nums_lc)}.')

# Generator expression of the same conditions
cubic_nums_gc = (i**3 for i in range(1500))
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 follwing generator produces infinite numbers.

In [ ]:

```
def infinite():
    count = 0
    while True:
        yield count
        count = count + 1

for i in infinite():
    print(i)
```

In [29]:

```
def generator(a):
    for i in range(a):
        yield i

gen = generator(6)
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
print(next(gen))
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]:

```
def square_number(num):
    for i in range(num):
        yield i**i

generator = square_number(6)

# Using 'while' loop
while True:
    try:
        print(f'The number using while loop is {next(generator)}.')
    except StopIteration:
        break

# Using 'for' loop
nlis = []
for square in square_number(6):
    nlis.append(square)
print(f'The numbers using for loop are {nlis}.')

# Using generator comprehension
square = (i**i for i in range(6))
square_list = []
square_list.append(next(square))
square_list.append(next(square))
square_list.append(next(square))
square_list.append(next(square))
square_list.append(next(square))
square_list.append(next(square))
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]:

```
import math
sum(i**i for i in range(6))
```

Out[42]:

```
3414
```

In [46]:

```
def fibonacci(numbers):
    a, b = 0, 1
    for _ in range(numbers):
        a, b = b, a+b
        yield a

def square(numbers):
    for i in numbers:
        yield i**2

print(sum(square(fibonacci(25))))
```

```
9107509825
```

# PYTHON
## FOR DATA SCIENCE
# CHEAT SHEET

created by
Tomi Mester

data36

I originally created this cheat sheet for my Python course and workshop participants.* But I have decided to open-source it and make it available **for everyone who wants to learn Python for data science**.

It's designed to give you a meaningful structure but also to let you add your own notes (that's why the empty boxes are there). **It starts from the absolute basics -** print('Hello World!') - and guides you to the intermediate level (for loops, if statements, importing advanced libraries). It also contains a few important functions of advanced libraries like pandas. **I added everything that you will need to get started as an absolute beginner** — and I'll continuously update and extend it to make it a full comprehensive cheat sheet for junior data analysts/scientists.

The ideal use case of this cheat sheet is that you print it in color and keep it next to you while you are learning and practicing Python on your computer.

Enjoy!

Cheers,
Tomi Mester

*The workshops and courses I mentioned:

**Online Python and Pandas tutorial (free):** data36.com/python-tutorial
**Python workshop for companies:** data36.com/python-workshop
**6-week Data Science course:** data36.com/jds

# VARIABLES IN PYTHON

In Python, you'll work with variables a lot.
You can assign a value to a variable as simply as:

```
variable_name = variable_value
```

If you assign a new value to a variable that you have used before, it will overwrite your previous value.

Examples:

```
a = 100
b = 'some_random_text'
c = True
d = 0.75
```

[your notes]

# BASIC DATA TYPES

In Python, we have quite a few different data types. But these four are the most important ones (for now):

1. **Integer.** A whole number without a fractional part. E.g. 100, 156, 2012412
2. **Float.** A number with a fractional part. E.g. 0.75, 3.1415, 961.1241250, 7/8
3. **Boolean.** A binary value. It can be either True or False.
4. **String.** It's a sequence of Unicode characters (e.g. numbers, letters, punctuation). It can be alphabetical letters only — or a mix of letters, numbers and other characters. In Python, it's easy to identify a string since it has to be between apostrophes (or quotation marks). E.g. 'hello', 'R2D2', 'Tomi', '124.56.128.41'

# ARITHMETIC OPERATORS

Let's assign two values!
**a = 3**
**b = 4**

The arithmetic operations you can do with them:

| Artihmetic operator | What does it do? | Result in our example |
|---|---|---|
| **a + b** | adds **a** to **b** | 7 |
| **a - b** | subtracts **b** from **a** | -1 |
| **a * b** | multiplies **a** by **b** | 12 |
| **a / b** | divides **a** by **b** | 0.75 |
| **b % a** | divides **b** by **a** and returns remainder | 1 |
| **a ** b** | raises **a** to the the power of **b** | 81 |

# DATA STRUCTURES

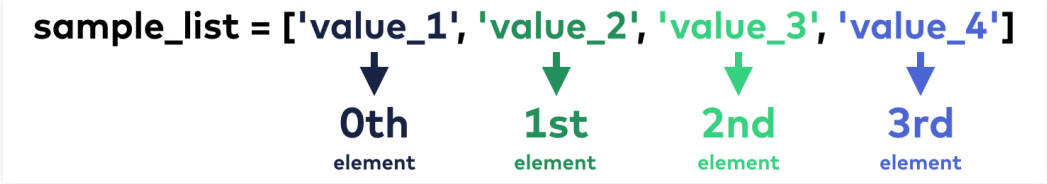
Data structures exist to organize your data and store related/similar data points in one "place." There are four data structure types. The two most important in data science are: **lists** and **dictionaries**.

## #1: LISTS

A list is a sequence of values. It can store integers, strings, booleans, anything - even a mix of these.
Example:
**sample_list = ['value1', 'value2', 'value3', 'value4', 1, 2, 3, 4, True, False]**

Querying an element of a list:
**sample_list[3]**

IMPORTANT! Python works with zero-based indexing. E.g.




Example:
**sample_list[3]** — (This returns 'value4'.)

## #2: DICTIONARY

A dictionary is a collection of key-value pairs. (Key is a unique identifier, value is the actual data.)

Example:

```
sample_dict = {'apple': 3,
               'tomato': 4,
               'orange': 1,
               'banana': 14,
               'is_store_open': True}
```

Querying an element of a dictionary:
**sample_dict['banana']** — (This returns **14**.)

## NESTED LISTS AND/OR DICTIONARY

You can create nested lists and dictionaries.

Example 1 (list within a list):

```
nested_list = ['val1', 'val2', ['nested_val1', 'nested_val2', 'nested_val3']]
```

Querying an element from the nested part:
**nested_list[2][0]** — (This returns **'nested_val1'**.)

Example 2 (list within a dictionary):

```
nested_dict = {'key_a': ['nested_val1', 'nested_val2', 'nested_val3'],
               'key_b': 'val2',
               'key_c': 'val3'}
```

Querying an element from the nested part:
**nested_dict['key_a'][0]** — (This returns **'nested_val1'**.)

## FUNCTIONS AND METHODS

You can run functions and methods on your Python objects. Most functions and methods are designed to perform a single action on your input and transform it into a (different) output.

Example:
**my_input = 'Hello'**
**len(my_input)**
Output: 5 (That's the number of characters in 'Hello'.)

Calling a Python function looks like this: **function_name(arguments)**
Calling a Python method looks like this: **input_value.method_name(arguments)**

More details on the difference between functions and methods:
https://data36.com/python-functions

## THE MOST IMPORTANT BUILT-IN FUNCTIONS

Let's assign a variable: **my_variable = 'Hello, World!'**

**print(my_variable)**
This prints the value of my_variable to the screen.
Output: **Hello, World!**

**len(my_variable)**
This returns the number of characters in a string - or the number of elements in a list. Output: **13** (That's the number of characters in 'Hello, World!')

**type(my_variable)**
This returns the data type of my_variable.
Output: **str** (That stands for string which is the data type of 'Hello, World!')

Find more Python functions here: https://data36.com/python-functions

## THE MOST IMPORTANT METHODS FOR PYTHON STRINGS

Let's assign a variable: **my_variable = 'Hello, World!'**

**my_variable.upper()**
This returns the uppercase version of a string.
Output: **'HELLO, WORLD!'**

**my_variable.lower()**
This returns the lowercase version of a string.
Output: **'hello, world!'**

**my_variable.split(',')**
This splits your string into a list. The argument specifies the separator that you want to use for the split.
Output: **['Hello', ' World']**

**my_variable.replace('World', 'Friend')**
This replaces a given string with another string. Note that it's case sensitive.
Output: **'Hello, Friend!'**

## THE MOST IMPORTANT METHODS FOR PYTHON LISTS

Let's make a list: **my_list = [10, 131, 351, 197, 10, 148, 705, 18]**

**my_list.append('new_element')**
It adds an element to the end of your list. The argument is the new element itself. This method updates your list and it doesn't have any output.

If you query the list after running this method:
**my_list**
Output: **[10, 131, 351, 197, 10, 148, 705, 18, 'new_element']**

**my_list.remove(10)**

It removes the first occurrence of the specified element from your list. This method updates your list and it doesn't have any output.

**my_list**
Output: **[131, 351, 197, 10, 148, 705, 18, 'new_element']**

**my_list.clear()**

It removes all elements of the list. This method updates your list and it doesn't have any output.

**my_list**
Output: **[]**

Find more Python functions and methods here:
https://data36.com/python-functions

All Python built-in functions:
https://docs.python.org/3/library/functions.html

All Python string methods:
https://docs.python.org/3/library/stdtypes.html#string-methods

All Python list methods:
https://docs.python.org/3/tutorial/datastructures.html

**IMPORTANT! These are only the built-in Python functions and methods. You can get access to many more with the import statement. (See page 12!)**

# IF STATEMENT

If statements are great for evaluating a condition and taking certain action(s) based on the result.

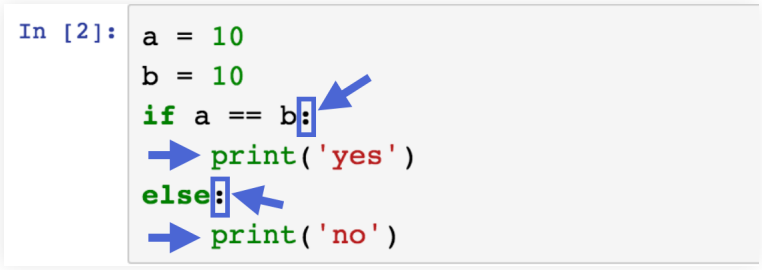
Example:

```
a = 10
b = 20
if a == b:
   print('yes')
else:
   print('no')
```

[your notes]

IMPORTANT! Be really careful with the syntax.

1. Never skip the colons at the end of the if and else lines!
2. Never skip the indentation (a tab or four spaces) at the beginning of the statement-lines!




In the if line (condition) you can use comparison and logical operators.
Let's see them. Assign four values!

**a = 3**
**b = 4**
**c = True**
**d = False**

| Comparison operator | What does it evaluate? | Result in our example |
|---|---|---|
| **a == b** | if **a** equals **b** | **False** |
| **a != b** | if **a** doesn't equal **b** | **True** |
| **a < b** | if **a** is less than **b** | **True** |
| **a > b** | if **a** is greater than **b** | **False** |

| Logical operator | What does it evaluate? | Result in our example |
|---|---|---|
| **c and d** | if both **c** and **d** are **True** | **False** |
| **c or d** | if either **c** or **d** is **True** | **True** |
| **not c** | returns the opposite of **c** | **False** |

## IF STATEMENT WITH MORE COMPLEX CONDITIONS

The condition can be complex.

Example:

```
a = 10
b = 20
c = 30
if (a + b) / c == 1 and c - b - a == 0:
   print('yes')
else:
   print('no')
```

## IF-ELIF-ELSE STATEMENT

You can use condition-sequences to evaluate multiple conditions.

Example:

```
a = 10
b = 11
c = 10
if a == b:
   print('a equals b, nice')
elif a == c:
   print('a equals c, nice')
else:
   print('a equals nothing... too bad')
```

Note: You can add as many elifs as you need.

# FOR LOOPS

For loops are for iterating through iterables (e.g. lists, strings, range() objects) and taking certain action(s) on the individual elements of the given iterable.

Example:

```
sample_list = ['value1', 'value2', 'value3', 'value4', 1, 2, 3, 4, True, False]
for i in sample_list:
    print(i)
```

Output:

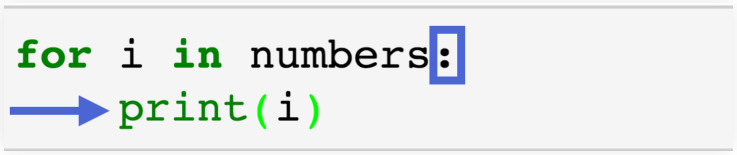
```
value1
value2
value3
value4
1
2
3
4
True
False
```

[your notes]

The action itself can be anything, not just **print()**. (Even multiple actions.)

IMPORTANT! Be really careful with the syntax.

1. Never skip the colons at the end of the for line!
2. Never skip the indentations (tabs or four spaces) in the body of the loop!



## FOR LOOPS (WITH range() OBJECTS)

If you want to iterate through numbers, you can use **range()**.

Example 1:

```
for i in range(5):
    print(i)
```

Output:

```
0
1
2
3
4
```

**range()** is a function. It accepts three (optional) arguments: start, stop, step.

Example 2:

```
for i in range(100,200,20):
    print(i)
```

Output:

```
100
120
140
160
180
```

More about for loops: https://data36.com/python-for-loops

## NESTED FOR LOOPS +
## FOR LOOPS AND IF STATEMENTS COMBINED

You can combine for loops with for loops (called nested for loops).
And you can combine for loops and if statements.

I wrote more about these here:
https://data36.com/python-nested

# PYTHON FORMATTING TIPS & BEST PRACTICES

## 1) ADD COMMENTS WITH THE # CHARACTER!

Example:

```
# This is a comment before my for loop.
for i in range(0, 100, 2):
    print(i)
```

## 2) VARIABLE NAMES

Conventionally, variable names should be written with lowercase letters, and the words in them separated by _ characters. Make sure that you choose meaningful and easy-to-distinguish variable names!

Example:

```
my_meaningful_variable_name = 100
```

## 3) USE BLANK LINES TO SEPARATE CODE BLOCKS VISUALLY!

Example:

In [7]:

```
down = 0
up = 100

for i in range(1,10):
    guessed_age = int((up+down)/2)
    answer = input('Are you ' + str(guessed_age) + " years old?")

    if answer == 'correct':
        print("Nice")
        break
    elif answer == 'less':
        up = guessed_age
    elif answer == 'more':
        down = guessed_age
    else:
        print('wrong answer')
```

## 4) USE WHITE SPACES AROUND OPERATORS AND ASSIGNMENTS!

Good example:

```
number_x = 10
number_y = 100
number_mult = number_x * number_y
```

Bad example:

```
number_x=10
number_y=100
number_mult=number_x*number_y
```

# IMPORTING OTHER PYTHON MODULES AND PACKAGES

Use the **import** statement to expand the original Python3 toolset with additional modules and packages.

General syntax:
**import [module_name]**

Or:
**from [module_name] import [item_name]**

# THE MOST IMPORTANT BUILT-IN MODULES FOR DATA SCIENTISTS

## RANDOM

Examples:

**import random**
This imports the random module. (No output.)

**random.random()**
This generates a random float between 0 and 1. (Output example: **0.6197724959**)

**random.randint(1,10)**
This generates a random integer between 1 and 10. (Output example: **4**)

## STATISTICS

Examples:

**import statistics**

This imports the statistics module.

**my_list = [0, 1, 1, 3, 4, 9, 15]**
**statistics.mean(my_list)**
**statistics.median(my_list)**
**statistics.mode(my_list)**
**statistics.stdev(my_list)**
**statistics.variance(my_list)**

These calculate the mean, median, mode, standard deviation and variance for the list called my_list. (Note: You have to run them one by one.)

## MATH

Examples:

**import math**

This imports the math module.

**math.factorial(5)**

This returns 5 factorial. (Output: **120**)

**math.pi**

This returns the value of pi. (Output: **3.141592653589793**)

**math.sqrt(5)**

This returns the square root of 5. (Output: **2.23606797749979**)

## DATETIME

Python3, by default, does not handle dates and times. But if you import the datetime module, you will get access to these functions, too.

Example:

**import datetime**

This imports the datetime module.

**datetime.datetime.now()**

This returns the current date and time in tuple format. (Note: A tuple is like a list, but can't be changed.)

Output: **datetime.datetime(2019, 7, 14, 0, 46, 30, 906311)**

**datetime.datetime.now().strftime("%F")**

This returns the current date and time in the usual yyyy-mm-dd format.

Output: **'2019-07-14'**

## CSV

This module helps you to open and manage .csv files in Python.

Example:

```
import csv
with open('example.csv') as csvfile:
   my_csv_file = csv.reader(csvfile, delimiter=';')
   for row in my_csv_file:
      print(row)
```

These few lines import the csv module and then open the example.csv file - where the fields are separated with semicolons (;). The last two lines of the code print all the rows (of the that .csv file we opened) one by one.

## MORE INFO ABOUT THE PYTHON BUILT-IN MODULES

- https://data36.com/python-import/
- https://docs.python.org/3/library/random.html
- https://docs.python.org/3/library/statistics.html
- https://docs.python.org/3/library/math.html
- https://docs.python.org/3/library/datetime.html
- https://docs.python.org/3/library/csv.html

## THE 5 MOST IMPORTANT "EXTERNAL" PYTHON LIBRARIES AND PACKAGES FOR DATA SCIENTISTS

- **Numpy**
- **Pandas**
- **Matplotlib**
- **Scikit-Learn**
- **Scipy**

# PANDAS

Pandas is one of the most popular Python libraries for data science and analytics. It helps you manage two-dimensional data tables and other data structures. It relies on Numpy, so when you import Pandas, you need to import Numpy first.

**import numpy as np**
**import pandas as pd**

# PANDAS DATA STRUCTURES

**Series:** Pandas Series is a one dimensional data structure ("a one dimensional ndarray") that can store values, with a unique index for each value.

```
In [4]: test_set_series
Out[4]: 0      15
        1      36
        2      41
        3      14
        4      69
        5      73
        6      92
        7      56
        8     101
        9     120
        10    175
        11    191
        12    215
        13    306
        14    241
        15    392
        dtype: int64
```

**DataFrame:** Pandas DataFrame is a two (or more) dimensional data structure – basically a table with rows and columns. The columns have names and the rows have indexes.

In [12]:
```
big_table
```

Out[12]:

| | user_id | phone_type | source | free | super |
|---|---|---|---|---|---|
| **0** | 1000001 | android | invite_a_friend | 5.0 | 0.0 |
| **1** | 1000002 | ios | invite_a_friend | 4.0 | 0.0 |
| **2** | 1000003 | error | invite_a_friend | 37.0 | 0.0 |
| **3** | 1000004 | error | invite_a_friend | 0.0 | 0.0 |
| **4** | 1000005 | ios | invite_a_friend | 6.0 | 0.0 |

## OPENING A .CSV FILE IN PANDAS

**pd.read_csv('/home/your/folder/file.csv', delimiter=';')**
This opens the .csv file that's located in /home/your/folder and called file.csv. The fields in the file are separated with semicolons (;).

**df = pd.read_csv('/home/your/folder/file.csv', delimiter=';')**
This opens a .csv file and stores the output into a variable called df. (The variable name can be anything else - not just df.)

**pd.read_csv('file.csv', delimiter=';', names = ['column1', 'column2', 'column3'])**
This opens file.csv. The fields in the file are separated with semicolons (;). We change the original names of the columns and set them to: 'column1', 'column2' and 'column3'.

## QUERYING DATA FROM PANDAS DATAFRAMES

**df**
It returns the whole dataframe. (Note: remember, when we opened the .csv file, we stored our dataframe into the **df** variable!)

**df.head()**
It returns the first 5 rows of **df**.

**df.tail()**
It returns the last 5 rows of **df**.

**df.sample(7)**
It returns 7 random rows from **df**.

**df[['column1', 'column2']]**
It returns column1 and column2 from **df**. (The output is in DataFrame format.)

**df.column1**
It returns column1 from **df**. (The output is in Series format.)

**df[my_dataframe.column1 == 'given_value']**
It returns all columns, but only those rows in which the value in column1 is 'given_value'. (The output is in DataFrame format.)

**df[['column1']][my_dataframe.column1 == 'given_value'].head()**
It takes the column1 column — and only those rows in which the value in column1 is 'given_value' — and returns only the first 5 rows. (The point is: you can combine things!)

## AGGREGATING IN PANDAS

The most important pandas aggregate functions:

- **.count()**
- **.sum()**
- **.mean()**
- **.median()**
- **.max()**
- **.min()**

Examples:
**df.count()**
It counts the number of rows in each column of **df**.

**df.max()**
It returns the maximum value from each column of **df**.

**df.column1.max()**
It returns the maximum value only from the column1 column of **df**.

# PANDAS GROUP BY

The **.groupby()** operation is usually used with an aggregate function (**.count()**, **.sum()**, **.mean()**, **.median()**, etc.). It groups the rows by a given column's values. (The column is specified as the argument of the **.groupby()** operation.) Then we can calculate the aggregate for each group and get that returned to the screen.

**df.groupby('column1').count()**
It counts the number of values in each column - for each group of unique column1 values.

**df.groupby('column1').sum()**
It sums the values in each column - for each group of unique column1 values.

**df.groupby('column1').min()**
It finds the minimum value in each column - for each group of unique column1 values.

**df.groupby('column1').max()**
It finds the maximum value in each column - for each group of unique column1 values.

# A FEW MORE USEFUL PANDAS METHODS

**df.merge(other_df)**
It joins df and **other_df** - for every row where the value of column1 from df equals the value of column1 from **other_df**.

**df.merge(other_df, how = 'inner', left_on = 'col2', right_on = 'col6')**
It joins **df** and **other_df** - for every row where the value of 'col2' from **df** ("left" table) equals the value of 'col6' from **other_df** ("right" table). The join type is an inner join.

**df.sort_values('column1')**
It returns every row and column from **df**, sorted by column1, in ascending order (by default).

**df.sort_values('column1', ascending = False)**
It returns every row and column from **df**, sorted by column1, in descending order.

**df.sort_values('column1', ascending = False).reset_index(drop = True)**
It returns every row and column from **df**, sorted by column1, in descending order. After sorting, it re-indexes the table: removes the old indexes and sets new ones.

**df.fillna('some_value')**
It finds all empty (NaN) values in **df** and replaces them with 'some_value'.

**Great pandas cheatsheet: https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf**

## CREATED BY

**Tomi Mester from Data36.com**

Tomi Mester is a data analyst and researcher. He's worked for Prezi, iZettle and several smaller companies as an analyst/consultant. He's the author of the Data36 blog where he writes posts and tutorials on a weekly basis about data science, AB-testing, online research and coding. He's an O'Reilly author and presenter at TEDxYouth, Barcelona E-commerce Summit, Stockholm Analytics Day and more.

## WHERE TO GO NEXT

**Find company workshops, online tutorials and online video courses on my website:** https://data36.com

Subscribe to my Newsletter list for useful stuff like this:
https://data36.com/newsletter

**Online Python and Pandas tutorial (free):** data36.com/python-tutorial
**Python workshop for companies:** data36.com/python-workshop
**6-week Data Science course:** data36.com/jds

# ⚡ Python Ultimate Guide ⚡

## 🔷 Fundamentals

├ 📌 Variables: x = 5

├ 📌 Print: print("Hello, World!")

├ 📌 Comments:

├ Single-line: # Comment

└ Multi-line: '''Comment'''

## 🔷 Data Types

├ 📌 Primitive:

├ String: "Hello"

├ Integer: 42

├ Float: 3.14

└ Boolean: True

├ 📌 Collections:

├ List: [1, 2, 3]

├ Tuple: (1, 2, 3)

├ Set: {1, 2, 3}

└ Dictionary: {"key": "value"}

## 🔷 Operators

├ 📌 Arithmetic: +, -, *, /, //, %, **

├ 📌 Comparison: ==, !=, <, >, <=, >=

├ 📌 Logical: and, or, not

├ 📌 Membership: in, not in

└ 📌 Identity: is, is not

## 🔷 Conditionals

├ 📌 If: `if x > y:`

├ 📌 Elif: `elif x < y:`

└ 📌 Else: `else:`

## 🔷 Loops

├ 📌 For: `for x in range(5):`

├ 📌 While: `while x < 5:`

├ 📌 Break: `break`

└ 📌 Continue: `continue`

## 🔷 Functions

├ 📌 Defining: `def my_function():`

├ 📌 Calling: `my_function()`

├ 📌 Default parameters: `def func(x, y=0):`

└ 📌 Variable-length arguments: `def func(*args, **kwargs):`

## 🔷 Classes & Objects

├ 📌 Class definition: `class MyClass:`

├ 📌 Constructor: `def __init__(self):`

├ 📌 Instance methods: `def method(self):`

├ 📌 Class variables: `class_var = 0`

├ 📌 Object instantiation: `my_object = MyClass()`

├ 📌 Inheritance: `class DerivedClass(BaseClass):`

└ 📌 Method overriding: `def method(self):`

## 🔷 Error Handling

├ 📌 Try: `try:`

├ 📌 Except: `except Exception as e:`

├ 📌 Raise: `raise ValueError("Error message")`

└ 📌 Finally: `finally:`

## 🔷 Importing Libraries

├ 📌 Import: import numpy

├ 📌 Alias: import numpy as np

└ 📌 Specific import: from math import pi

## 🔷 File I/O

├ 📌 Open: with open("file.txt", "r") as file:

├ 📌 Read: file.read()

├ 📌 Write: with open("file.txt", "w") as file:

├ 📌 Append: with open("file.txt", "a") as file:

## 🔷 List Comprehensions

├ 📌 Syntax: [expression for item in iterable if condition]

## 🔷 Lambda Functions

├ 📌 Syntax: lambda arguments: expression

## 🔷 Iterators & Generators

├ 📌 Iterator: iter(obj)

├ 📌 Next item: next(iterator)

├ 📌 Generator function: def my_generator(): yield value

└ 📌 Generator expression: (expression for item in iterable if condition)

## 🔷 Context Managers

├ 📌 Defining: class MyContext:

├ 📌 Enter method: def __enter__(self):

├ 📌 Exit method: def __exit__(self, exc_type, exc_value, traceback):

└ 📌 Using: with MyContext() as my_context:

## 🔹 Built-in Functions

├ 📏 len(obj) → Length of object

├ 🧮 sum(iterable[, start]) → Sum of elements

├ 🔝 max(iterable[, key]) → Maximum element

├ 🔙 min(iterable[, key]) → Minimum element

├ 📁 sorted(iterable[, key][, reverse]) → Sorted list

├ 🎯 range(stop[, start][, step]) → Sequence of numbers

├ 🤝 zip(*iterables) → Iterator of tuples

├ 🗺️ map(function, iterable) → Apply function to all items

├ 🧹 filter(function, iterable) → Filter elements by function

└ 🎓 isinstance(obj, classinfo) → Check object's class

## 🔷 String Methods

├ 🔡 lower() → Lowercase

├ 🔠 upper() → Uppercase

├ strip([chars]) → Remove leading/trailing characters

├ ✂️ split([sep][, maxsplit]) → Split by separator

├ ↔️ replace(old, new[, count]) → Replace substring

├ 🔎 find(sub[, start][, end]) → Find substring index

└ 🎨 format(*args, **kwargs) → Format string

## 🔷 List Methods

├ ➕ append(item) → Add item to end

├ 📬 extend(iterable) → Add elements of iterable

├ 👇 insert(index, item) → Insert item at index

├ ❌ remove(item) → Remove first occurrence

├ 🍭 pop([index]) → Remove & return item

├ 🔢 index(item[, start][, end]) → Find item index

├ 🧮 count(item) → Count occurrences

├ 📊 sort([key][, reverse]) → Sort list

└ 🔃 reverse() → Reverse list

## Dictionary Methods

- keys() → View list of keys
- values() → View list of values
- items() → View key-value pairs
- get(key[, default]) → Get value for key
- update([other]) → Update dictionary
- pop(key[, default]) → Remove & return value
- clear() → Remove all items

## Set Methods

- add(item) → Add item
- update(iterable) → Add elements of iterable
- discard(item) → Remove item if present
- remove(item) → Remove item or raise KeyError
- pop() → Remove & return item
- clear() → Remove all items
- union(*others) → Union of sets
- intersection(*others) → Intersection of sets
- difference(*others) → Difference of sets
- issubset(other) → Check if subset
- issuperset(other) → Check if superset

## Regular Expressions

- import re
- re.search(pattern, string)
- re.match(pattern, string)
- re.findall(pattern, string)
- re.sub(pattern, repl, string)
- Common patterns:

- \d: Digit
- \w: Word character
- \s: Whitespace
- .: Any character (except newline)
- ^: Start of string
- $: End of string
- *: Zero or more repetitions
- +: One or more repetitions
- ?: Zero or one repetition
- {n}: Exactly n repetitions
- {n,}: At least n repetitions
- {,m}: At most m repetitions
- {n,m}: Between n and m repetitions (inclusive)

## Decorators

- Defining: def my_decorator(func):
- Applying: @my_decorator

## Modules & Packages

- Creating a module: Save as .py file
- Importing a module: import my_module
- Creating a package: Create directory with __init__.py
- Importing from a package: from my_package import my_module

## Virtual Environments

- Creating: python -m venv myenv
- Activating:
- Windows: myenv\Scripts\activate
- Unix/Mac: source myenv/bin/activate
- Deactivating: deactivate

## Package Management (pip)

- Install: pip install package_name
- Uninstall: pip uninstall package_name
- Upgrade: pip install --upgrade package_name
- List installed packages: pip list
- Show package details: pip show package_name

## Date & Time

- import datetime
- Current date & time: datetime.datetime.now()
- Date object: datetime.date(year, month, day)
- Time object: datetime.time(hour, minute, second, microsecond)
- Format: datetime.datetime.strftime(format)
- Parse: datetime.datetime.strptime(date_string, format)
- Common format codes: %Y, %m, %d, %H, %M, %S

## JSON

- import json
- JSON to Python: json.loads(json_string)
- Python to JSON: json.dumps(obj)
- Read from file: json.load(file)
- Write to file: json.dump(obj, file)

## Threading

- import threading
- Create a thread: t = threading.Thread(target=function, args=(arg1, arg2))
- Start a thread: t.start()
- Wait for thread to finish: t.join()

## 🔷 Multiprocessing

├ 📥 import multiprocessing

├ ⚙️ Create a process: p = multiprocessing.Process(target=function, args=(arg1, arg2))

├ 🚀 Start a process: p.start()

└ ⏸️ Wait for process to finish: p.join()

## 🔷 Working with Databases (SQLite)

├ 📥 import sqlite3

├ 🗄️ Connect to a database: conn = sqlite3.connect('mydb.sqlite')

├ 📍 Cursor object: cursor = conn.cursor()

├ 📜 Execute SQL commands: cursor.execute("CREATE TABLE my_table (id INTEGER, name TEXT)")

├ 💾 Commit changes: conn.commit()

├ 📚 Fetch results: cursor.fetchall()

└ 🚪 Close the connection: conn.close()

## 🔷 Web Scraping (BeautifulSoup)

├ 📥 from bs4 import BeautifulSoup

├ 🍲 Create a BeautifulSoup object: soup = BeautifulSoup(html_content, 'html.parser')

├ 🔍 Find elements by tag: soup.find_all('tag_name')

├ 🏷️ Access element attributes: element['attribute_name']

└ 📝 Get element text: element.text

## 🔷 Web Requests (Requests)

├ 📥 import requests

├ 📩 GET request: response = requests.get(url)

├ 📤 POST request: response = requests.post(url, data=payload)

├ 📄 Response content: response.content

├ 🧩 JSON response: response.json()

└ 🚦 Response status code: response.status_code

## 🔷 Web Development (Flask)

├ 📥 from flask import Flask, render_template, request, redirect, url_for

├ 🌐 Create a Flask app: app = Flask(__name__)

├ 🚩 Define a route: @app.route('/path', methods=['GET', 'POST'])

├ 🏃 Run the app: app.run(debug=True)

├ 💌 Return a response: return "Hello, World!"

├ 🎨 Render a template: return render_template('template.html', variable=value)

├ 📩 Access request data: request.form['input_name']

└ 🔀 Redirect to another route: return redirect(url_for('route_function'))

## 🔷 Data Science Libraries

├ 🧪 NumPy: import numpy as np

├ 📊 pandas: import pandas as pd

├ 🎨 Matplotlib: import matplotlib.pyplot as plt

├ 📈 seaborn: import seaborn as sns

├ 😈 scikit-learn: import sklearn

├ 🧠 TensorFlow: import tensorflow as tf

├ 🧬 Keras: from tensorflow import keras

├ 🔥 PyTorch: import torch

## 🔷 Command Line Arguments (argparse)

├ 📥 import argparse

├ 📋 Create an ArgumentParser: parser = argparse.ArgumentParser(description='Description of your program')

├ 📑 Add arguments: parser.add_argument('--arg_name', type=str, help='Description of the argument')

├ 🧩 Parse arguments: args = parser.parse_args()

└ 📌 Access argument values: args.arg_name

🔷 Logging

├ 📥 import logging

├ ⚙️ Basic configuration: logging.basicConfig(level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s')

├ 📊 Logging levels: logging.debug(), logging.info(), logging.warning(), logging.error(), logging.critical()

## 🔷 Environment Variables

├ 📥 import os

├ 🌐 Get an environment variable: os.environ.get('VAR_NAME')

└ 📝 Set an environment variable: os.environ['VAR_NAME'] = 'value'

## 🔷 Type Hints

├ 📥 from typing import List, Dict, Tuple, Optional, Union, Any

├ 🏷️ Function type hints: def my_function(param: int, optional_param: Optional[str] = None) -> List[int]:

└ 📌 Variable type hints: my_variable: Dict[str, int] = {}