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!)
```

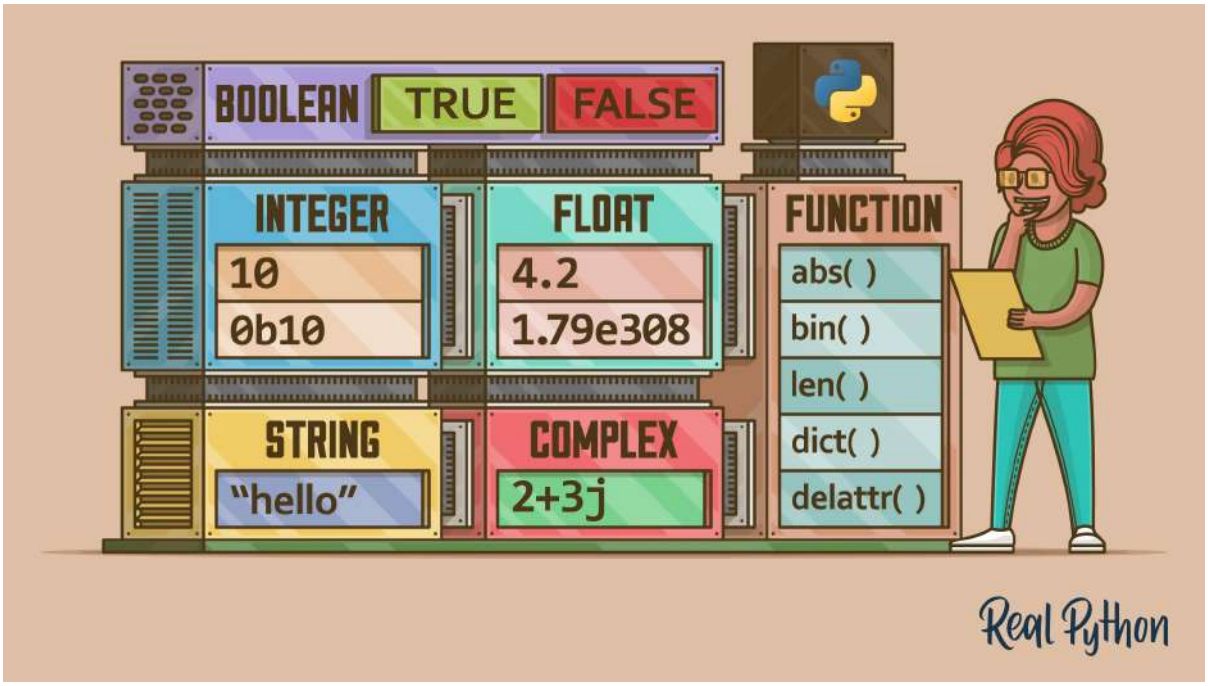
```
  File "C:\Users\test\AppData\Local\Temp/ipykernel_13804/974508531.py", line 2
    print('Hello, World!)
          ^
SyntaxError: unterminated string literal (detected at line 2)
```

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]:

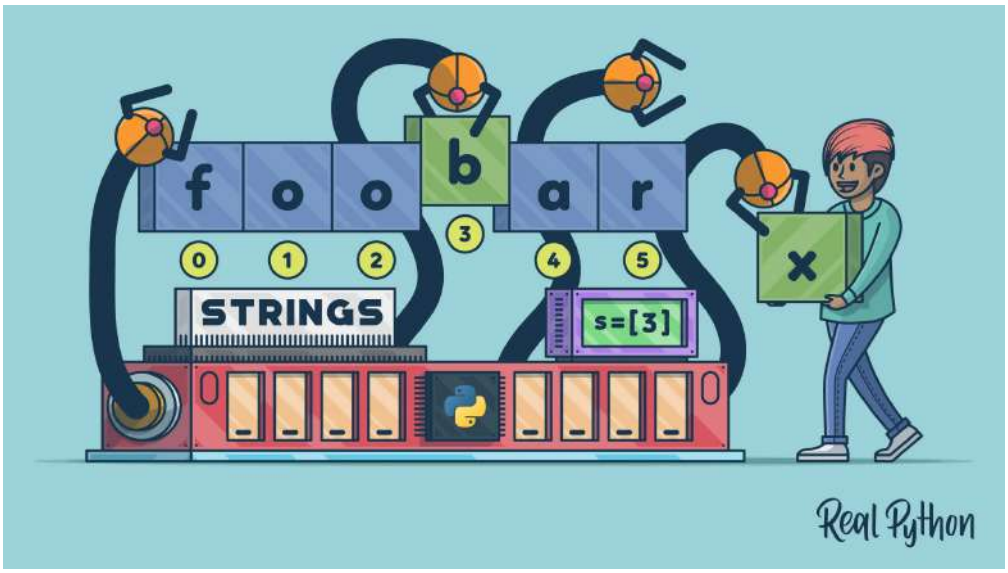
```
x, y, z = 8, 4, 2      # the values of x, y, and z can be written in one line.
print(x, y, z)
print(x)
print(y)
print(z)
print(x/y)
print(x/z)
print(y/z)
print(x+y+z)
print(x*y*z)
print(x-y-z)
print(x/y/z)
print(x//y//z)
print(x%y%z)

```

```
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))
```

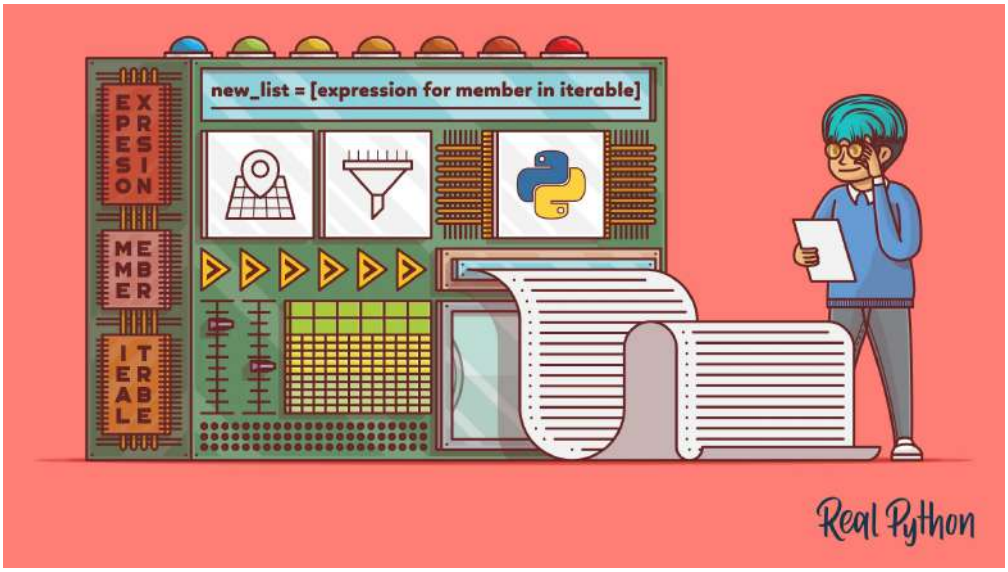
```
Hello World!
Hi, My name is Bob and I am 36 years old.
Hi, My name is Bob and I am 36 years old.
Hi, My name is Bob and I am 36 years old.
```

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
```

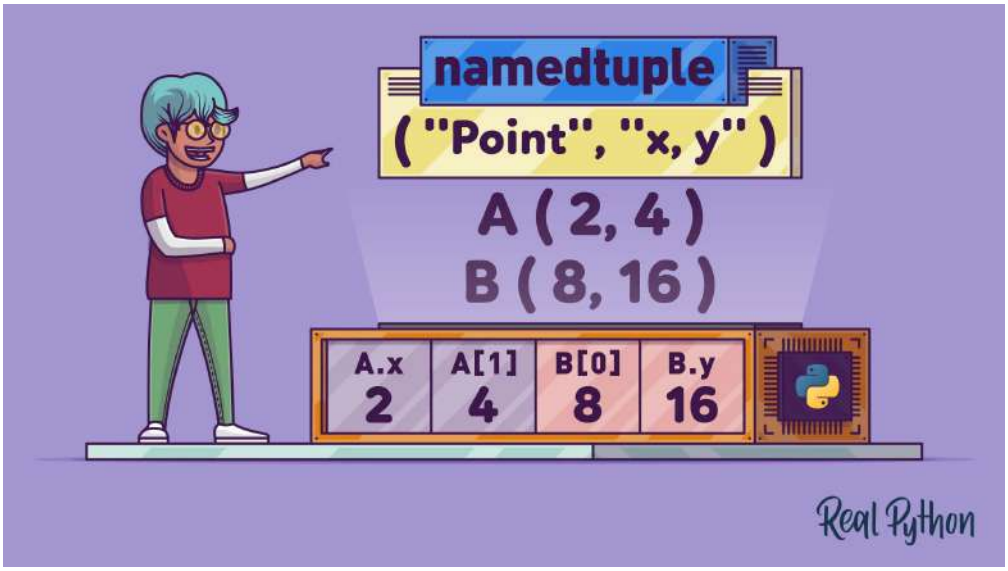
Python Tutorial

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
```

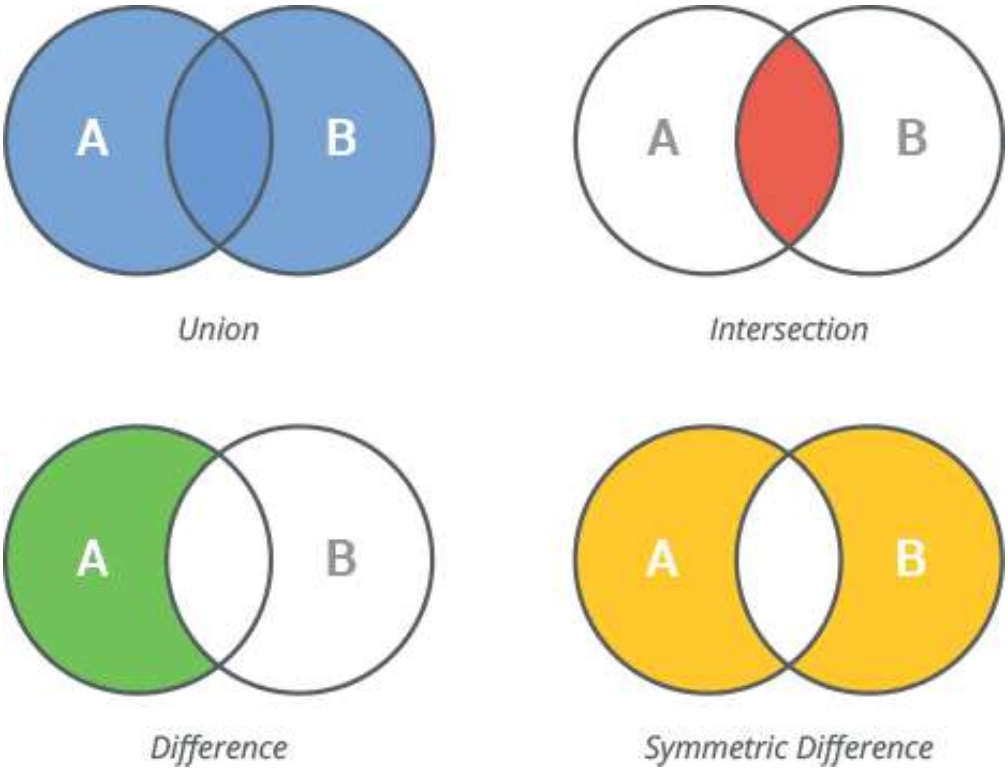
```
(0,)
<class 'tuple'>
```

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)
```

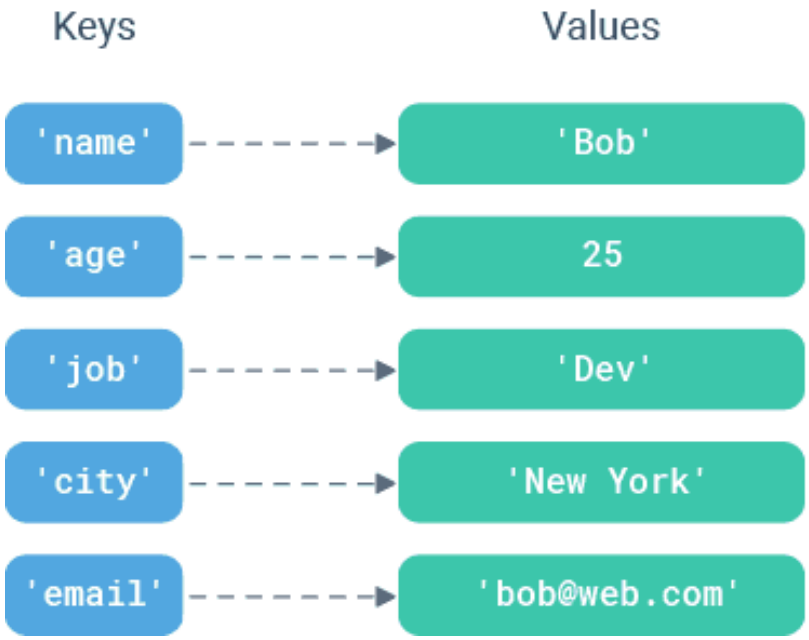
```
{0, 1, 2, 3, 34, 5, 8, 13, 21}
{1, 2, 3, 34, 5, 8, 13, 21}
```

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]:

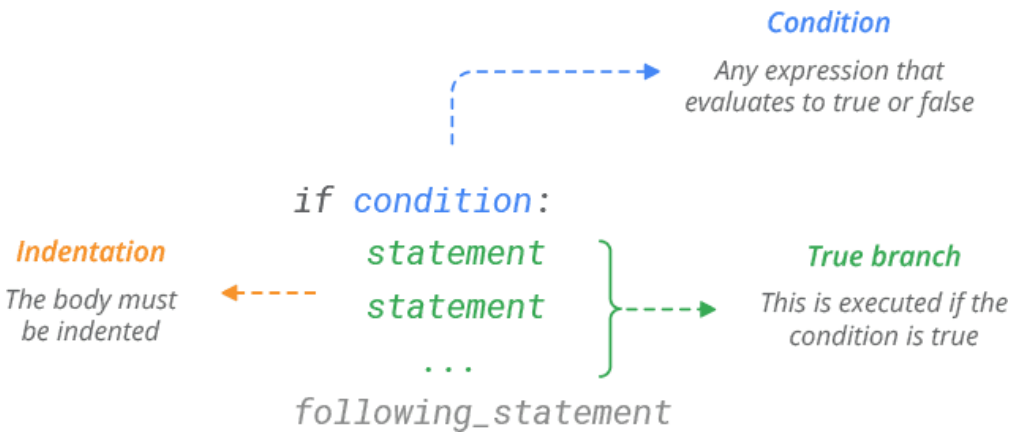
```
# Compare strings
print('Hello' == 'Python')
print('Hello' != 'Python')
print('Hello' <= 'Python')
print('Hello' >= 'Python')
print('Hello' < 'Python')
print('Hello' > 'Python')
print('B'>'A')  # According to ASCII table, the values of A and B are equal 65 and 66, respectively.
print('a'>'b')  # 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!')
```

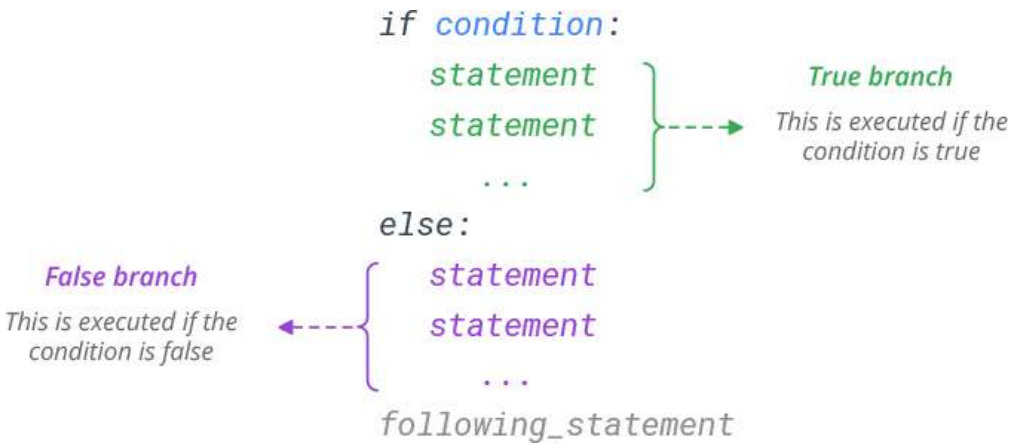
Hello, python!

In [5]:

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

This is true.

else statement

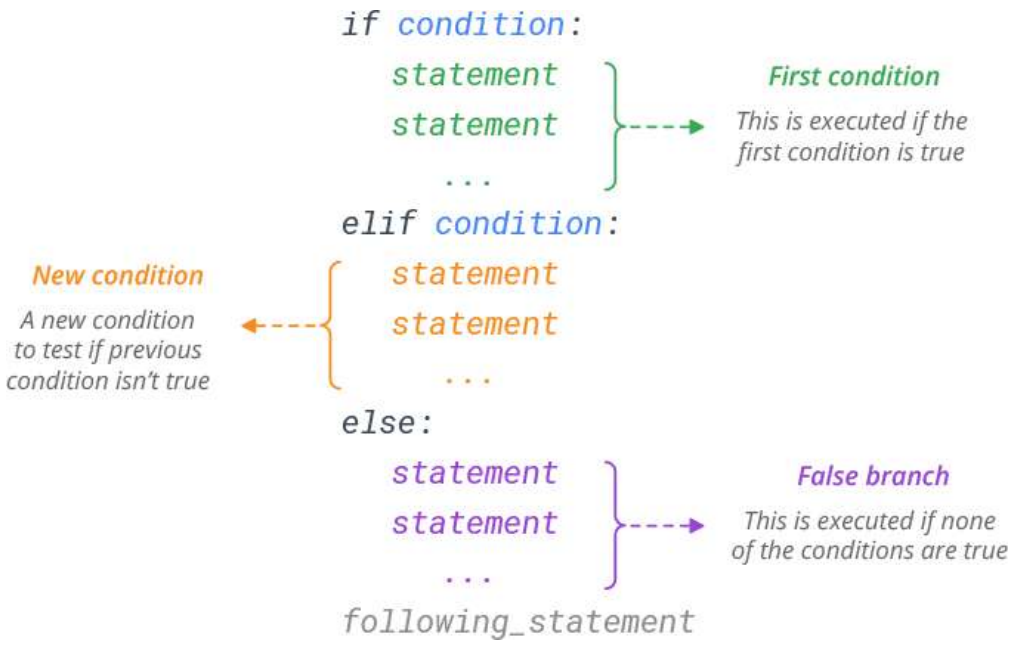


In [8]:

```
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.

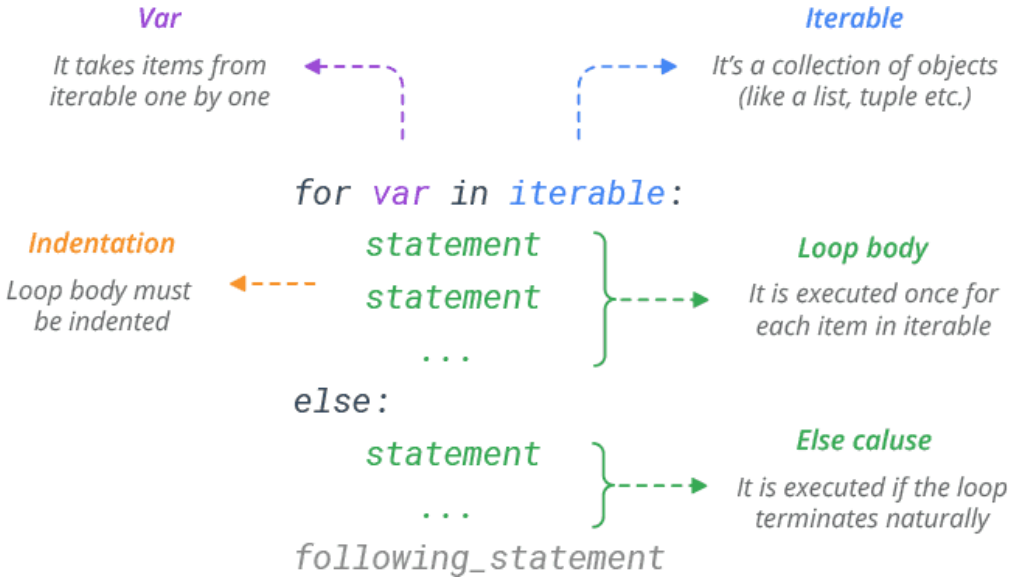
In [3]:

```
# 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.
"""
```

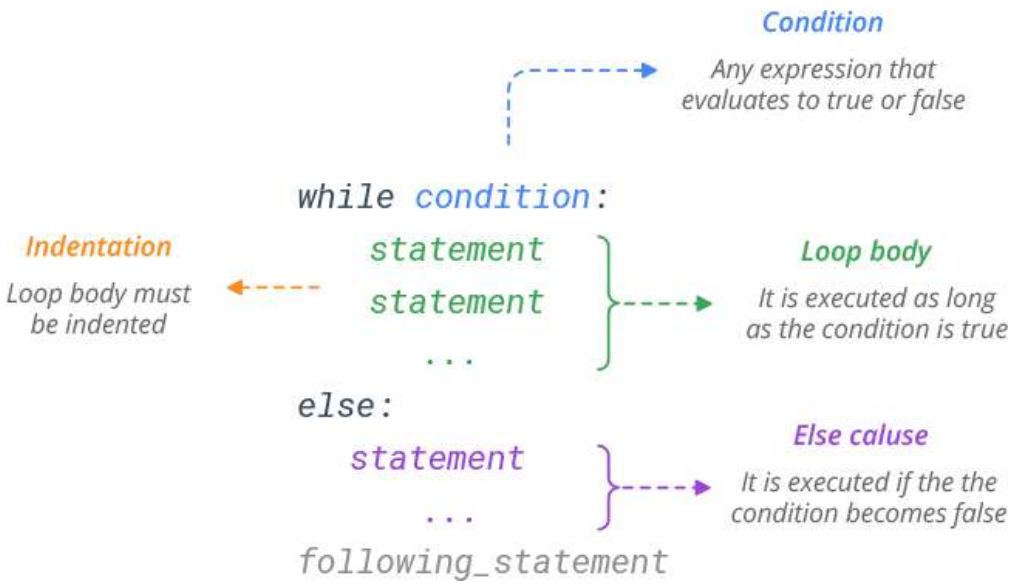
```
1
2
4
```

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
```

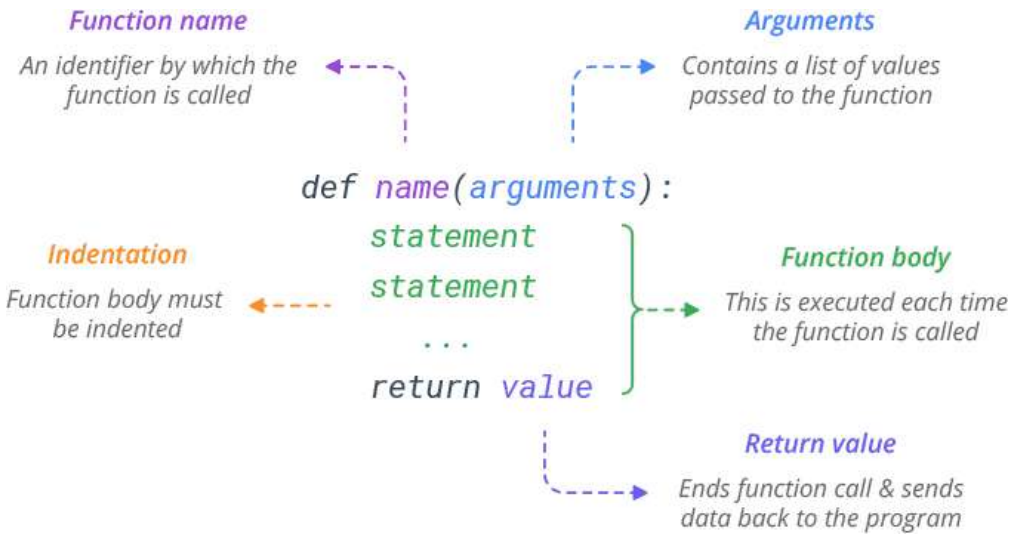
```
0
1
2
```

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)
```

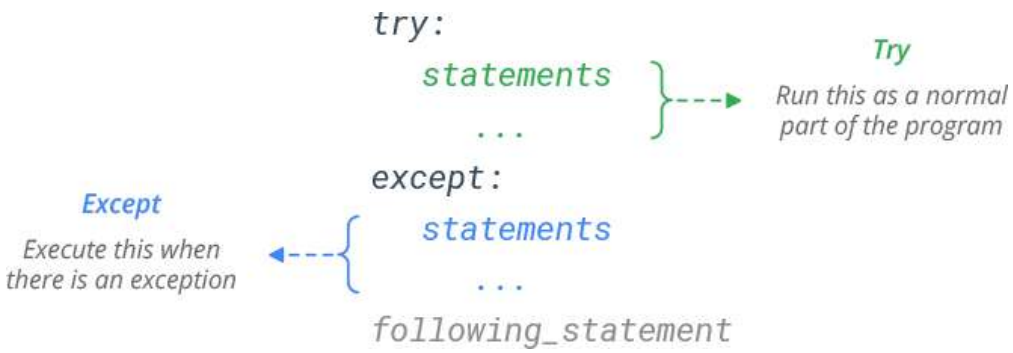
This code gives us a KeyError.

```
---------------------------------------------------------------------------
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

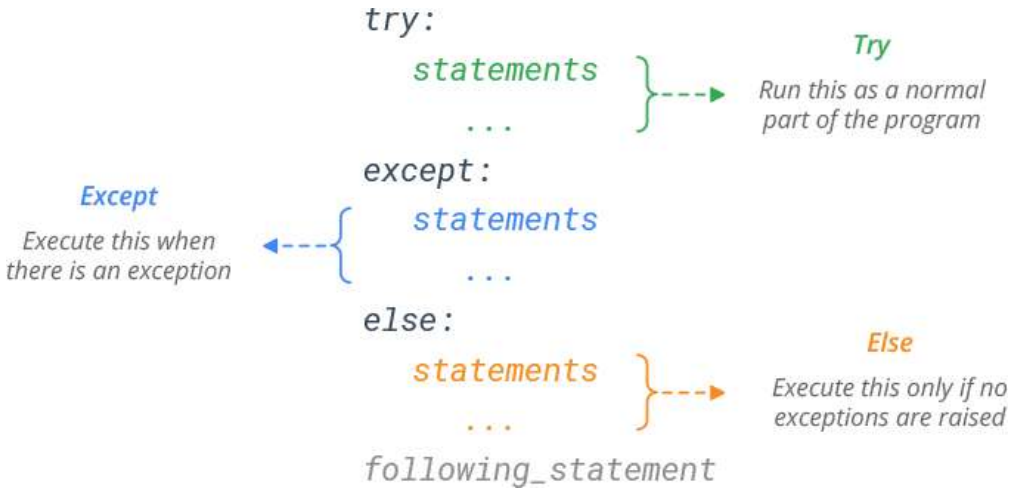
try/except/except etc.

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

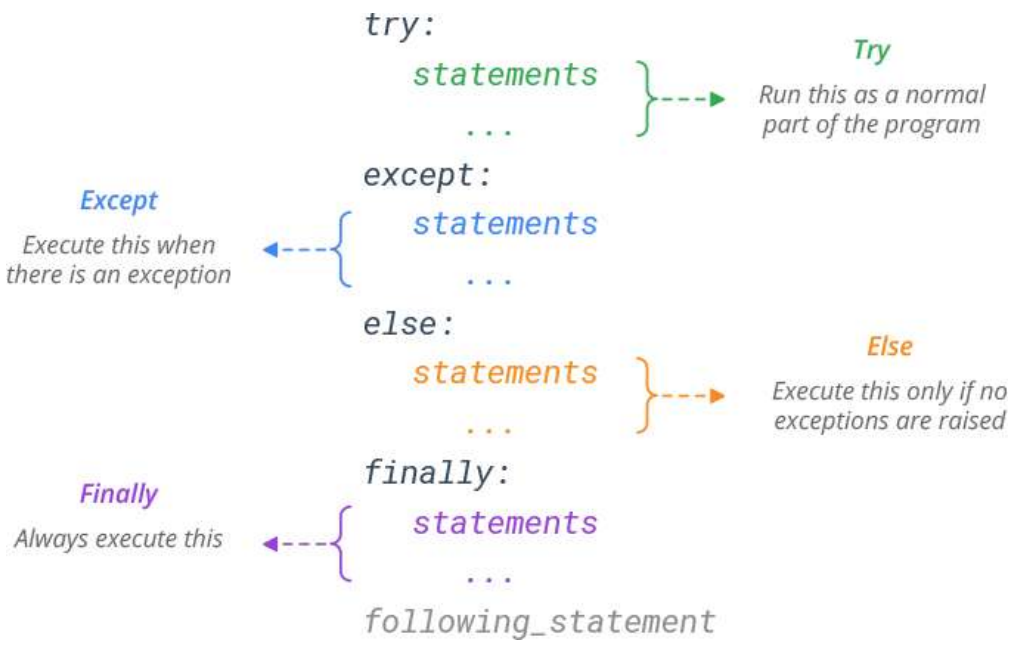


In [3]:

```
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.
```