Python Data Fundamentals

1. Variable Assignment
2. Data Types
3. Data containers in Python
4. Index and Slicing
5. List methods
6. Data Builtin Fucntions in Python
7. String Formatting
8. String methods

Variable Assignment

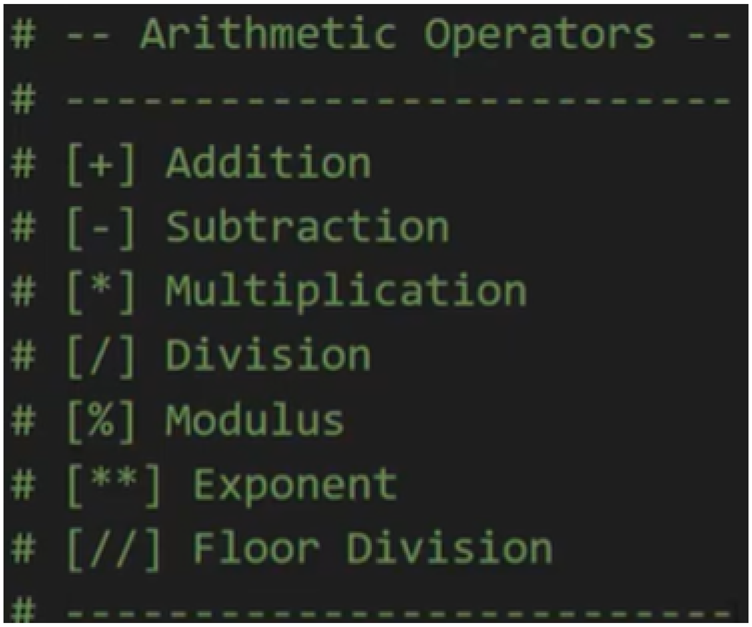
Regular mathematical operators (+, -, *, / and power) can be performed in code cells.

In [1]:

```
# This is a code cell.
# Type your code in a cell and run it. For example
(10 * 2 + 5 - 4) / 3
```

Out[1]:

```
7.0
```



```
# -- Arithmetic Operators --
# -------------------------
# [+] Addition
# [-] Subtraction
# [*] Multiplication
# [/] Division
# [%] Modulus
# [**] Exponent
# [//] Floor Division
# -------------------------
```

In [2]:

```
# Power in python:
10**2
```

Out[2]:

```
100
```

The results from the above operations can not be reused in subsequent codes because they were not assigned any name. To use them elsewhere will require to assign an operation to a variable. For example

```
In [3]: x = (10 * 2 + 5 - 4) / 3
        y = x**2 + 2*x - 5
```

To write an output to the screen, we can type the **variable name or use the print() function** .

```
In [4]: x

Out[4]: 7.0
```

```
In [5]: y

Out[5]: 58.0
```

```
In [6]: # This will only output the last variable in the cell
        x
        y

Out[6]: 58.0
```

```
In [7]: # This will print the values of x and y
        print(x)
        print(y)

        7.0
        58.0
```

Consider the values of x and y that we printed in the last example. It will be more intuitive to include some descriptions to the outputs. We can use a string to include some descriptions in the **print** function.

```
In [8]: print("x = ", x)
        print("y = ", y)

        x =  7.0
        y =  58.0
```

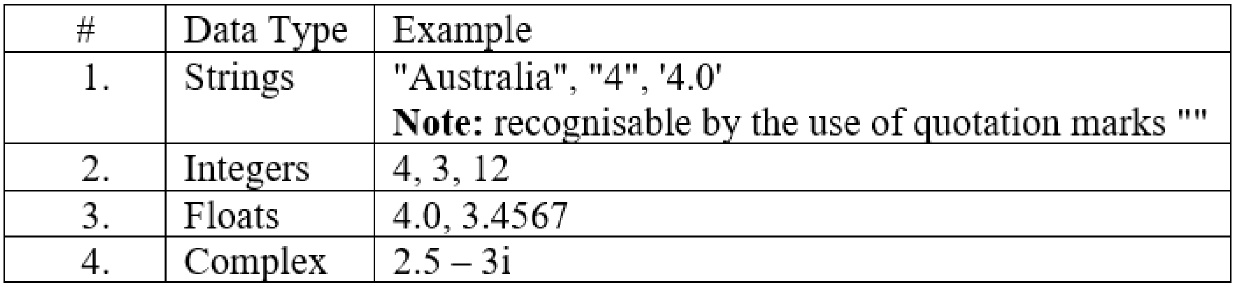
```
In [9]: x, y, z= 1,2,3
```

```
In [10]: x

Out[10]: 1
```

```
In [11]: x,y,z

Out[11]: (1, 2, 3)
```

Data Types

The following common data types are recognised by python:



#	Data Type	Example
1.	Strings	"Australia", "4", '4.0' **Note:** recognisable by the use of quotation marks ""
2.	Integers	4, 3, 12
3.	Floats	4.0, 3.4567
4.	Complex	2.5 – 3i

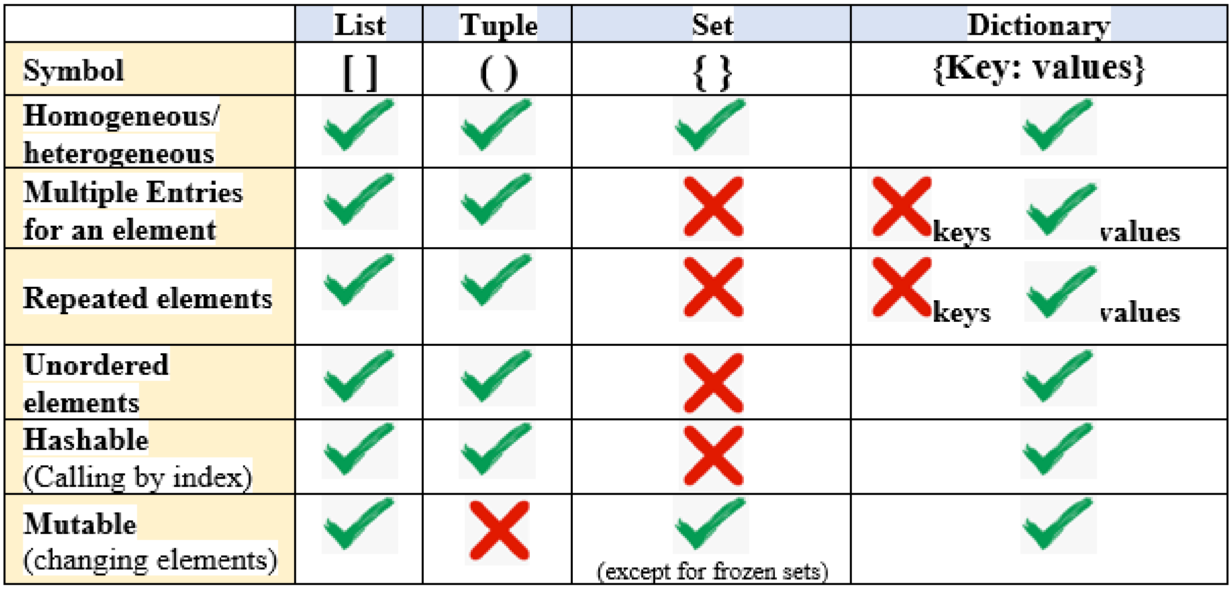
```
In [12]: type("Australia")

Out[12]: str

In [13]: type("4")

Out[13]: str

In [14]: type(4)

Out[14]: int

In [15]: type(3.14567)

Out[15]: float
```

Converting data types

Now, it is important to note that functions exist to convert from one data type to another. For example, to **convert an integer 5 to string, we can invoke the function str() and to convert string '5.0' to float, we can use the float() function.**

```
In [16]: my_Float=3.14567
         my_Float

Out[16]: 3.14567

In [17]: int(my_Float)

Out[17]: 3

In [18]: str(my_Float)

Out[18]: '3.14567'

In [19]: float(3)

Out[19]: 3.0
```

Data containers in Python



	List	Tuple	Set	Dictionary
Symbol	[ ]	( )	{ }	{Key: values}
Homogeneous/ heterogeneous	✓	✓	✓	✓
Multiple Entries for an element	✓	✓	✗	✗keys ✓values
Repeated elements	✓	✓	✗	✗keys ✓values
Unordered elements	✓	✓	✗	✓
Hashable (Calling by index)	✓	✓	✗	✓
Mutable (changing elements)	✓	✗	✓ (except for frozen sets)	✓

List [ ]

```
In [20]:  # Define a list: Mehtod 1:
          myList=[1,2,3]
          myList

Out[20]:  [1, 2, 3]
```

```
In [21]:  # Define a list: Mehtod 2:
          myList=list((1,2,3))
          myList

Out[21]:  [1, 2, 3]
```

```
In [22]:  #   Homogeneous/ heterogeneous: Y
          #   Multiple Entries for an element: Y
          #   Unordered: Y
          myList1=[1,3,2,2,"Jh",[5,7,6,9],True]
          myList1

Out[22]:  [1, 3, 2, 2, 'Jh', [5, 7, 6, 9], True]
```

```
In [23]:  # Hashability: Y
          myList1[4]

Out[23]:  'Jh'
```

```
In [24]:  myList1[5][3]

Out[24]:  9
```

```
In [25]:  myList1[0:4]

Out[25]:  [1, 3, 2, 2]
```

```
In [26]:  myList1[1:6:3]
```

```
Out[26]: [3, 'Jh']
```

```
In [27]: myList_2d=[[1,2,3],[4,5,6],[7,8,9]]
         myList_2d
```

```
Out[27]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
In [28]: myList_2d[0]
```

```
Out[28]: [1, 2, 3]
```

```
In [29]: myList_2d[1][2]
```

```
Out[29]: 6
```

```
In [30]: # Mutability: Y
         myList1=[1,3,2,2,"Jh",[5,7]]
         myList1[2]="New-Value"
         myList1
```

```
Out[30]: [1, 3, 'New-Value', 2, 'Jh', [5, 7]]
```

```
In [31]: myList1[0:2]=[99,999,9999]
         myList1
```

```
Out[31]: [99, 999, 9999, 'New-Value', 2, 'Jh', [5, 7]]
```

```
In [32]: myList1[0:2]=[99]
         myList1
```

```
Out[32]: [99, 9999, 'New-Value', 2, 'Jh', [5, 7]]
```

Tuple ( )

```
In [33]: # Define a tuple: Mehtod 1:
         myTuple=(1,2,3)
         myTuple
```

```
Out[33]: (1, 2, 3)
```

```
In [34]: # Define a tuple: Mehtod 2:
         myTuple=tuple((1,2,3))
         myTuple
```

```
Out[34]: (1, 2, 3)
```

```
In [35]: #   Homogeneous/ heterogeneous: Y
         #   Multiple Entries for an element: Y
         #   Order/unordered: Y
         myTuple1=(1,3,2,2,"Jh",[5,7])
         myTuple1
```

```
Out[35]: (1, 3, 2, 2, 'Jh', [5, 7])
```

In [36]:

```
# Hashability: : y
myTuple1[2]
```

Out[36]:

```
2
```

In [37]:

```
myTuple1[5][1]
```

Out[37]:

```
7
```

In [38]:

```
# Mutability: N
#myTuple1[2]="New-Value" # This will results in error
#myTuple1
```

Set { }

In [39]:

```
# Define a set: Mehtod 1:
mySet={2,3,1}
mySet
```

Out[39]:

```
{1, 2, 3}
```

In [40]:

```
# Define a set: Mehtod 2:
mySet=set((2,3,1))
mySet
```

Out[40]:

```
{1, 2, 3}
```

In [41]:

```
#   Homogeneous/ heterogeneous: Y
#   Multiple Entries for an element: N
#   Unordered : N,
#   Repetetion : N
mySet1={1,3,2,2,"Jh"}
mySet1
```

Out[41]:

```
{1, 2, 3, 'Jh'}
```

In [42]:

```
# Example
myset2=set([1, 3, 'New-Value', 2,2, 'Jh'])
myset2
```

Out[42]:

```
{1, 2, 3, 'Jh', 'New-Value'}
```

In [43]:

```
# Hashability: : N
#mySet1[2]           # This will results in error
```

In [44]:

```
# Mutability: Y
#mySet1[2]="New-Value"
#mySet1       # This will results in error
```

The error doesn't mean that sets are immutable. It is still related to the fact that sets are unhashable. To show that sets are mutable (unless frozen), let's apply the **remove & add** methods to achieve the same result we wanted

In [45]:

```
mySet1.add(999)
```

```
mySet1
```

```
Out[45]: {1, 2, 3, 999, 'Jh'}
```

```
In [46]: # freeze set
         # let's define a frozen set
         myFrozenSet=frozenset(mySet1)
         myFrozenSet
```

```
Out[46]: frozenset({1, 2, 3, 999, 'Jh'})
```

```
In [47]: #myFrozenSet.remove(3.75)  # This will result in error
```

```
In [48]: #myFrozenSet.add('kkk')  # This will result in error
```

This simply shows that frozen sets are immutable.

```
In [49]: # How to unfreez a frozenset!
         # Rui: I could not find a function that can unfreez a frozenset. May be it is somewhere
         # Yet below way works fine.
         Unfrozenset=set(list(myFrozenSet))
         Unfrozenset
```

```
Out[49]: {1, 2, 3, 999, 'Jh'}
```

```
In [50]: Unfrozenset.add("Rui Huang")
         Unfrozenset
```

```
Out[50]: {1, 2, 3, 999, 'Jh', 'Rui Huang'}
```

Dictionery {key:value}

```
In [51]: # Define a dictionery: Mehtod 1:
         myDict1={"Jhon":36,"Archer":25,"Charlie":40}
         myDict1
```

```
Out[51]: {'Jhon': 36, 'Archer': 25, 'Charlie': 40}
```

```
In [52]: # Define a dictionery: Mehtod 2:
         name=["Jhon","Archer","Charlie"]
         age=[30,20,35]
         myDict2=dict(zip(name,age))
         myDict2
```

```
Out[52]: {'Jhon': 30, 'Archer': 20, 'Charlie': 35}
```

Each of the dictionery attributes can be converted to list if necessary as illustrated below:

```
In [53]: # Dic_keys
         Names=list(myDict2.keys())
         Names
```

```
Out[53]: ['Jhon', 'Archer', 'Charlie']
```

```
In [54]: # Dic_values
```

```
age=list(myDict2.values())
age
```

Out[54]:
```
[30, 20, 35]
```

In [55]:
```
#   Homogeneous/ heterogeneous: Y
#   Multiple Entries for a key: Y
#   Unordered : Y
#   Values repetetion: Y
myDict3={"Jam":1,"Fam":[5,50], "Dam":(1,2,2), 3:"Sam"}
myDict3
```

Out[55]:
```
{'Jam': 1, 'Fam': [5, 50], 'Dam': (1, 2, 2), 3: 'Sam'}
```

In [56]:
```
# Hashablity: : Y
# Dictionary_name['key']=value
myDict3['Jam']
```

Out[56]:
```
1
```

In [57]:
```
myDict3['Fam']
```

Out[57]:
```
[5, 50]
```

In [58]:
```
myDict3['Fam'][0]
```

Out[58]:
```
5
```

In [59]:
```
# Mutablity: Y
# 1 changing a value
# Dictionary_name['key'] = new_value
myDict3['Jam']=1000
myDict3
```

Out[59]:
```
{'Jam': 1000, 'Fam': [5, 50], 'Dam': (1, 2, 2), 3: 'Sam'}
```

In [60]:
```
myDict3['Fam'][0]=5000
myDict3
```

Out[60]:
```
{'Jam': 1000, 'Fam': [5000, 50], 'Dam': (1, 2, 2), 3: 'Sam'}
```

In [61]:
```
# Mutablity: Y
# 2 adding a new value
# Dictionary_name['new_key'] = new_value
myDict3['Arther']="100"
myDict3
```

Out[61]:
```
{'Jam': 1000, 'Fam': [5000, 50], 'Dam': (1, 2, 2), 3: 'Sam', 'Arther': '100'}
```

In [62]:
```
# Can we add a new value to key's values
# Answer: As Michaels suggested, yes we can append a value to kes's values.
myDict3['Fam'].append("Michaels")
myDict3
```

Out[62]:
```
{'Jam': 1000,
 'Fam': [5000, 50, 'Michaels'],
 'Dam': (1, 2, 2),
 3: 'Sam',
 'Arther': '100'}
```

Data continer conversion

```
In [63]: myList = [ 3, 2, 1,'C', 'U', 2, 'N','C','C']
         myList
```

```
Out[63]: [3, 2, 1, 'C', 'U', 2, 'N', 'C', 'C']
```

```
In [64]: myTuple = tuple(myList)
         myTuple
```

```
Out[64]: (3, 2, 1, 'C', 'U', 2, 'N', 'C', 'C')
```

```
In [65]: mySet = set(myList)
         mySet
```

```
Out[65]: {1, 2, 3, 'C', 'N', 'U'}
```

```
In [66]: myDict=dict(zip(myList,myList))
         myDict
```

```
Out[66]: {3: 3, 2: 2, 1: 1, 'C': 'C', 'U': 'U', 'N': 'N'}
```

Data containers Unpacking

List [ ]

```
In [5]: my_list=[1,2,3]
        my_list
```

```
Out[5]: [1, 2, 3]
```

```
In [7]: a,b,c=my_list
```

```
In [8]: a
```

```
Out[8]: 1
```

```
In [9]: a,c
```

```
Out[9]: (1, 3)
```

Tuple ( )

```
In [31]: my_tuple=[1,2,3]
         my_tuple
```

```
Out[31]: [1, 2, 3]
```

```
In [32]: a,b,c=my_tuple
In [33]: a

Out[33]: 1
```

```
In [34]: a,c

Out[34]: (1, 3)
```

Dictionery {key:value}

keys

```
In [35]: dic={"Sam": 1, "Bam": 2, "Fam":3}
```

```
In [36]: a,b,c=dic
```

```
In [37]: a

Out[37]: 'Sam'
```

```
In [17]: a,c

Out[17]: ('Sam', 'Fam')
```

values

```
In [19]: a,b,c=dic.values()
```

```
In [20]: a

Out[20]: 1
```

```
In [21]: a,c

Out[21]: (1, 3)
```

items

```
In [22]: a,b,c=dic.items()
```

```
In [23]: a

Out[23]: ('Sam', 1)
```

```
In [24]: a,c

Out[24]: (('Sam', 1), ('Fam', 3))
```

String

```
In [27]: string="good"
In [28]: a,b,c,d=string
```

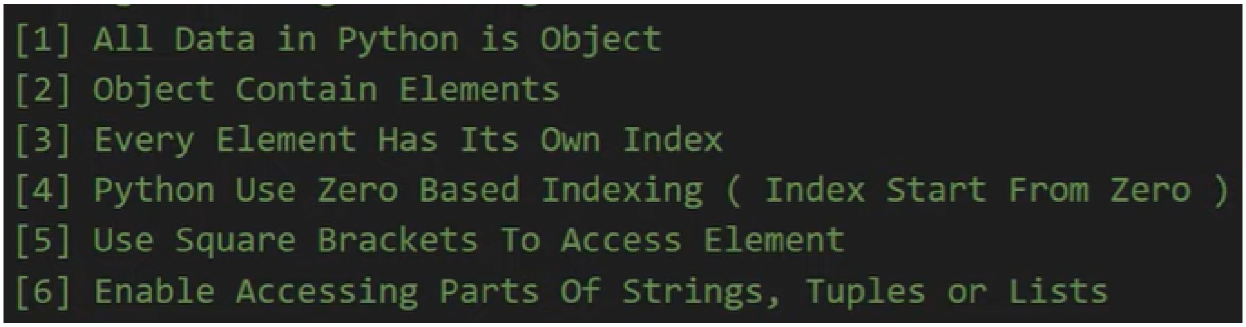
```
In [29]: a
```

```
Out[29]: 'g'
```

```
In [30]: a,c
```

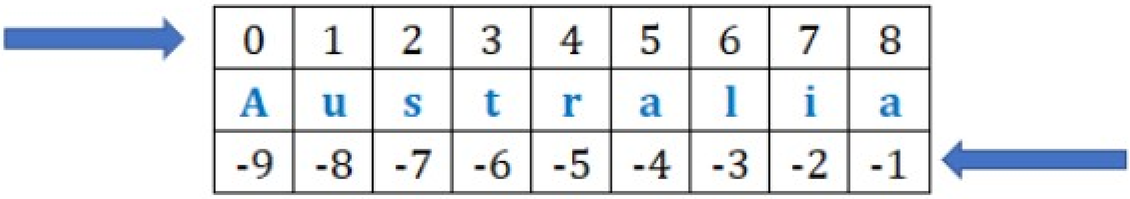
```
Out[30]: ('g', 'o')
```

Index and Slicing



```
[1] All Data in Python is Object
[2] Object Contain Elements
[3] Every Element Has Its Own Index
[4] Python Use Zero Based Indexing ( Index Start From Zero )
[5] Use Square Brackets To Access Element
[6] Enable Accessing Parts Of Strings, Tuples or Lists
```

Python indexing typically goes from left to right and starts at 0 in unit steps. Right to left indexing is also allowed and starts at -1 in steps of -1 as illustrated below:



0	1	2	3	4	5	6	7	8
A	u	s	t	r	a	l	i	a
-9	-8	-7	-6	-5	-4	-3	-2	-1

Slicing involves obtaining a subset or subsets of a string at specified locations (indexes). </font>

```
In [67]: # Example 1
         myString="Australia"
```

```
In [68]: # Let's get the first letter in myString
         myString[0]
```

```
Out[68]: 'A'
```

```
In [69]: # to get the last letter (or the first letter from right)
         myString[-1]
```

```
Out[69]: 'a'
```

Slicing: **name[start:stop:steps]**.

In [70]:

```
# Slicing: [Start:End: Steps]
myString[0:5:1]
```

Out[70]:

```
'Austr'
```

In [71]:

```
myString[0:5]
```

Out[71]:

```
'Austr'
```

In [72]:

```
myString[:5]
```

Out[72]:

```
'Austr'
```

In [73]:

```
myString[5:]
```

Out[73]:

```
'alia'
```

In [74]:

```
myString[-1:-10:-1]
```

Out[74]:

```
'ailartsuA'
```

In [75]:

```
# Example 2
myList=[1,3,4,5,6,7,8,9,10]
myList
```

Out[75]:

```
[1, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [76]:

```
myList[0]
```

Out[76]:

```
1
```

In [77]:

```
myList[-1]
```

Out[77]:

```
10
```

In [78]:

```
myList[0:5:3]
```

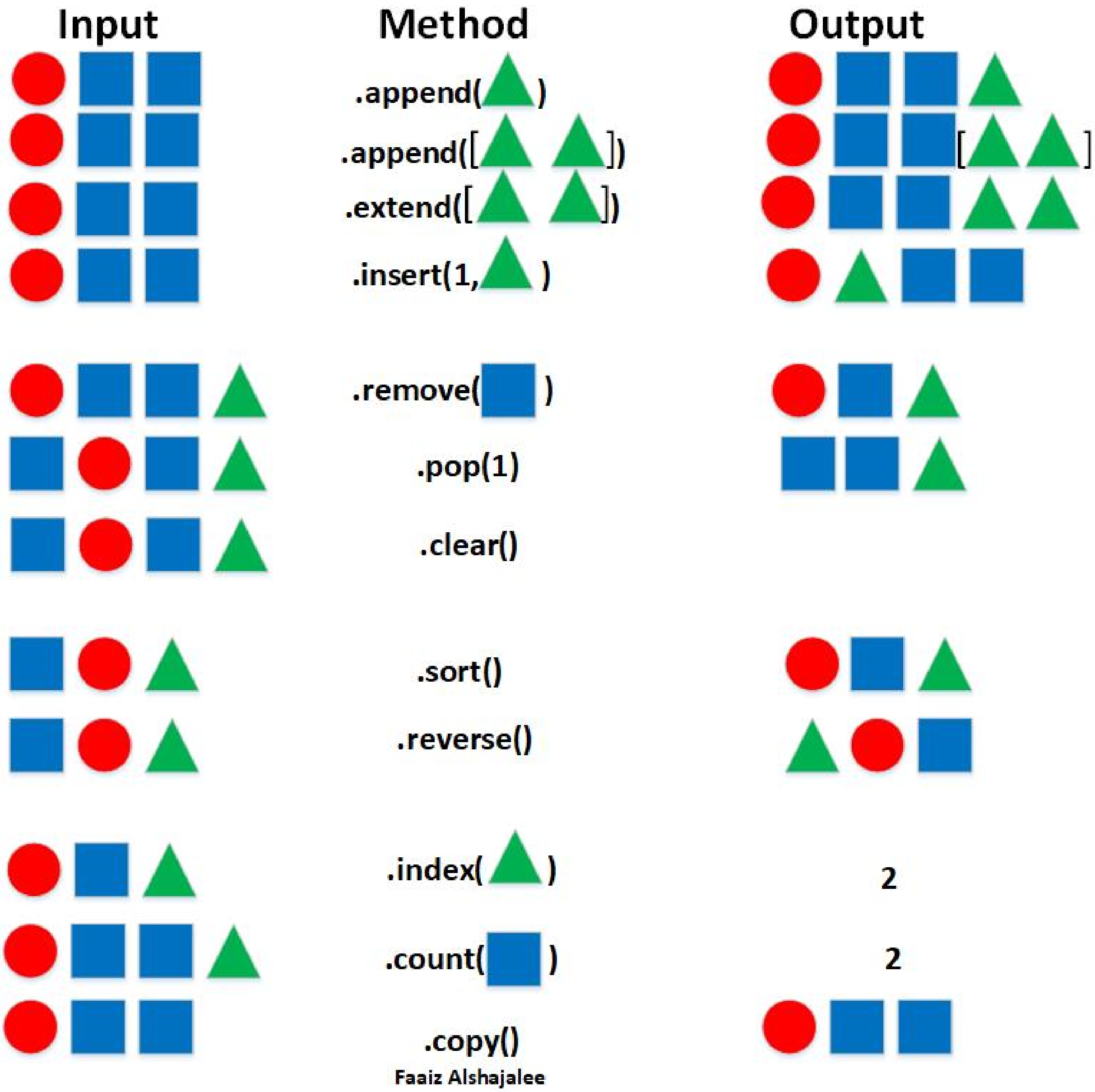
Out[78]:

```
[1, 5]
```

List container methods

Each data container has a number of useful methods that facilitate its use in programming. The help() function can be used to obtain the list of methods available for a given data container

Visualization of some Python methods



Append method

```
In [79]: myList=[1,2,3]
         myList
```

```
Out[79]: [1, 2, 3]
```

```
In [80]: myList.append(4)
         myList
```

```
Out[80]: [1, 2, 3, 4]
```

```
In [81]: myList=[1,2,3]
         myList.append([4,5])
         myList
```

```
Out[81]: [1, 2, 3, [4, 5]]
```

Extend method

```
In [82]: myList=[1,2,3]
         myList.extend([4,5])
         myList
```

```
Out[82]: [1, 2, 3, 4, 5]
```

Insert method

```
In [83]: myList=[1,2,3]
         myList.insert(1,10)
         myList
```

```
Out[83]: [1, 10, 2, 3]
```

```
In [84]: myList=[1,2,3]
         myList.insert(1,[4, 5])
         myList
```

```
Out[84]: [1, [4, 5], 2, 3]
```

Remove method

```
In [85]: myList=[1,2,2,3,2]
         myList.remove(2)
         myList
```

```
Out[85]: [1, 2, 3, 2]
```

Pop method

```
In [86]: myList=[1,2,3]
         myList.pop(1)
         myList
```

```
Out[86]: [1, 3]
```

Clear method

```
In [87]: myList=[1,2,3]
         myList.clear()
         myList
```

```
Out[87]: []
```

Sort method

```
In [88]: myList=[3,1,2]
         myList.sort()
         myList
```

```
Out[88]: [1, 2, 3]
```

Reverse method

```
In [89]: myList=[3,1,2]
         myList.reverse()
         myList
```

```
Out[89]: [2, 1, 3]
```

```
In [90]: # sort & reverse: Method 1
         myList=[3,1,2]
         myList.sort(reverse=True)
         myList
```

```
Out[90]: [3, 2, 1]
```

```
In [91]: # sort & reverse: Method 2
         myList=[3,1,2]
         myList.sort()
         myList.reverse()
         myList
```

```
Out[91]: [3, 2, 1]
```

Index method

```
In [92]: myList=[1,2,3,"HHH"]
         myList.index("HHH")
```

```
Out[92]: 3
```

Count method

```
In [93]: myList=[1,2,2,3]
         myList.count(2)
```

```
Out[93]: 2
```

Copy method

```
In [94]: myList=[1,2,3]
         myList_copy=myList
         myList_copy.remove(2)
         myList_copy
```

```
Out[94]: [1, 3]
```

```
In [95]: myList
```

```
Out[95]: [1, 3]
```

You observe that myList has also changed. This because setting myList_copy equal to myList did not create a new copy but rather referencing the original list. To make a distinct copy of a list, we need to use the copy method.

In [96]:

```
# Now let's make a copy of myList
myList=[1,2,3]
myList_copy=myList.copy()
myList_copy.remove(2)
myList_copy
```

Out[96]:

```
[1, 3]
```

In [97]:

```
myList
```

Out[97]:

```
[1, 2, 3]
```

In [ ]:

Data Builtin Fucntions in Python

Help( )

In [98]:

```
help(list)
```

```
Help on class list in module builtins:

class list(object)
 |  list(iterable=(), /)
 |  
 |  Built-in mutable sequence.
 |  
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |  
 |  Methods defined here:
 |  
 |  __add__(self, value, /)
 |      Return self+value.
 |  
 |  __contains__(self, key, /)
 |      Return key in self.
 |  
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |  
 |  __eq__(self, value, /)
 |      Return self==value.
 |  
 |  __ge__(self, value, /)
 |      Return self>=value.
 |  
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |  
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |  
 |  __gt__(self, value, /)
 |      Return self>value.
 |  
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |  
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |  
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |  
 |  __iter__(self, /)
 |      Implement iter(self).
 |  
 |  __le__(self, value, /)
 |      Return self<=value.
 |  
 |  __len__(self, /)
 |      Return len(self).
 |  
 |  __lt__(self, value, /)
 |      Return self<value.
 |  
 |  __mul__(self, value, /)
 |      Return self*value.
 |  
 |  __ne__(self, value, /)
 |      Return self!=value.
 |  
 |  __repr__(self, /)
 |      Return repr(self).
 |  
 |  __reversed__(self, /)
 |      Return a reverse iterator over the list.
 |  
 |  __rmul__(self, value, /)
 |      Return value*self.
 |  
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |  
 |  __sizeof__(self, /)
 |      Return the size of the list in memory, in bytes.
 |  
 |  append(self, object, /)
 |      Append object to the end of the list.
 |  
 |  clear(self, /)
 |      Remove all items from list.
 |  
 |  copy(self, /)
 |      Return a shallow copy of the list.
 |  
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |  
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |  
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |      
 |      Raises ValueError if the value is not present.
 |  
 |  insert(self, index, object, /)
 |      Insert object before index.
 |  
 |  pop(self, index=-1, /)
 |      Remove and return item at index (default last).
 |      
 |      Raises IndexError if list is empty or index is out of range.
 |  
 |  remove(self, value, /)
 |      Remove first occurrence of value.
 |      
 |      Raises ValueError if the value is not present.
 |  
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |  
 |  sort(self, /, *, key=None, reverse=False)
 |      Sort the list in ascending order and return None.
 |      
 |      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
 |      order of two equal elements is maintained).
 |      
 |      If a key function is given, apply it once to each list item and sort them,
 |      ascending or descending, according to their function values.
 |      
 |      The reverse flag can be set to sort in descending order.
 |  
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |  
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |  
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |  
 |  __hash__ = None
```

In [99]:

```
#help(set)
```

Type( )

In [100…

```
my_num=[1,2,3]
type(my_num)
```

Out[100…

```
list
```

In [101…

```
type((1,2,3))
```

Out[101…

```
tuple
```

In [102…

```
type({2,3,1})
```

Out[102…

```
set
```

In [103…

```
type({"Jhon":36,"Archer":25})
```

Out[103…

```
dict
```

Len( )

This is used to obtain the number of elements in a data container

In [104…

```
myList
```

Out[104…

```
[1, 2, 3]
```

In [105…

```
len(myList)
```

Out[105…

```
3
```

In [106…

```
len("Australia")
```

Out[106…

```
9
```

Range( )

Range is an immutable sequence of integers.

It is defined using the function **range(start, stop, step)**. </font>

In [107…

```
# Let's define a range as follows
myRange = range(0, 50, 5)
myRange
```

Out[107…

```
range(0, 50, 5)
```

The output does not have much meaning. Let's use the list function to obtain a clearer output.

In [108…

```
# Let's convert the previous output to a list
myRange = list(range(0, 5, 1))
myRange
```

Out[108…

```
[0, 1, 2, 3, 4]
```

In [26]:

```
range(10)
```

Out[26]:

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

Slice( )

slice(start, stop, step)

In [56]:

```
a = ["A", "B", "C", "D", "E", "F"]

print(a[:5])
print(a[slice(5)])

print("\n"*2)
print(a[2:5])
print(a[slice(2, 5)])
```

```
['A', 'B', 'C', 'D', 'E']
['A', 'B', 'C', 'D', 'E']



['C', 'D', 'E']
['C', 'D', 'E']
```

Sum( )

In [14]:

```
# sum(iterable, start)
a = [1, 10, 9, 30]

print(sum(a))
print(sum(a, 40))
```

```
50
90
```

Min( )

min(item, item , item, or iterator)

In [1]:

```
print(min(1, 10, -50, 20, 30))

myNumbers = (1, 20, -50, -100, 100)
print(min(myNumbers))

print(min("X", "Z", "Osama"))
```

```
-50
-100
Osama
```

Max( )

max(item, item , item, or iterator)

In [47]:

```
print(max(1, 10, -50, 20, 30))

myNumbers = (1, 20, -50, -100, 100)
print(max(myNumbers))

print(max("X", "Z", "Osama"))
```

```
30
100
Z
```

Map( )

[1] Map Take A Function + Iterator: **map (Function, Iterator)**. </font>

[2] Map Called Map Because It Map The Function On Every Element

[3] The Function Can Be Pre-Defined Function or Lambda Function

In [81]:
```
# Use Map With Predefined Function

def formatText(text):
  return text.strip().capitalize()
```

In [83]:
```
myTexts = [" OSama ", "AHMED", "  sAYed  "]

myFormatedData = list(map(formatText, myTexts))

myFormatedData
```

Out[83]:
```
['Osama', 'Ahmed', 'Sayed']
```

In [69]:
```
for name in list(map(formatText, myTexts)):

  print(name)
```

```
Osama
Ahmed
Sayed
```

In [80]:
```
# Use Map With Lambda Function
myTexts = [" OSama ", "AHMED", "  sAYed  "]

for name in list(map((lambda text: text.strip().capitalize()), myTexts)):

  print(name)
```

```
Osama
Ahmed
Sayed
```

Filter( )

[1] Filter Take A Function + Iterator **filter (Function, Iterator)**. </font>

[2] Filter Run A Function On Every Element

[3] The Function Can Be Pre-Defined Function or Lambda Function

[4] The Function Need To Return Boolean Value

[5] Filter Out All Elements For Which The Function Return True

In [87]:
```
# Example 1

def checkNumber(num):

  return num > 10

myNumbers = [0, 0, 1, 19, 10, 20, 100, 5, 0]

myResult = filter(checkNumber, myNumbers)
```

```
list(myResult)
```

Out[87]:
```
[19, 20, 100]
```

In [93]:
```
# Example 2

def checkName(name):
  return name.startswith("O")

myTexts = ["Osama", "Omer", "Ahmed", "Sayed", "Othman"]
myReturnedData = filter(checkName, myTexts)
list(myReturnedData)
```

Out[93]:
```
['Osama', 'Omer', 'Othman']
```

In [91]:
```
"Omer".startswith("O")
```

Out[91]:
```
True
```

In [92]:
```
"Ahmed".startswith("O")
```

Out[92]:
```
False
```

In [ ]:
```
# Lambda & Filter
# Example 3:
num=[1,2,3,4,5,6,7,8,9,10]
even_num= list(filter(lambda x: x%2==0,num))
even_num
```

In [4]:
```
# Example 4: Filter and Lambda

myNames = ["Osama", "Omer", "Omar", "Ahmed", "Sayed", "Othman", "Ameer"]

for p in filter(lambda name: name.startswith("A"), myNames):
  print(p)
```

```
Ahmed
Ameer
```

Reduce( )

[1] Reduce Take A Function + Iterator **filter (Function, Iterator)**. </font>

[2] Reduce Run A Function On FIrst and Second Element And Give Result

[3] Then Run Function On Result And Third Element

[4] Then Run Function On Rsult And Fourth Element And So On

[5] Till One ELement is Left And This is The Result of The Reduce

[6] The Function Can Be Pre-Defined Function or Lambda Function

In [95]:
```
((((1 + 8) + 2) + 9) + 100)
```

Out[95]:
```
120
```

In [96]:
```python
from functools import reduce

def sumAll(num1, num2):

  return num1 + num2

numbers = [1, 8, 2, 9, 100]

reduce(sumAll, numbers)
```

Out[96]:
```
120
```

In [98]:
```python
reduce(lambda num1, num2: num1 + num2, numbers)
```

Out[98]:
```
120
```

Enumerate( )

enumerate(iterable, start=0)

In [103…
```python
mySkills = ["Html", "Css", "Js", "PHP"]

mySkillsWithCounter = enumerate(mySkills)

for skill in mySkillsWithCounter:
    print(skill)
```

```
(0, 'Html')
(1, 'Css')
(2, 'Js')
(3, 'PHP')
```

In [104…
```python
mySkills = ["Html", "Css", "Js", "PHP"]

mySkillsWithCounter = enumerate(mySkills,10)

for skill in mySkillsWithCounter:
    print(skill)
```

```
(10, 'Html')
(11, 'Css')
(12, 'Js')
(13, 'PHP')
```

In [113…
```python
mySkills = ["Html", "Css", "Js", "PHP"]

mySkillsWithCounter = enumerate(mySkills)

for counter, skill in mySkillsWithCounter:

    print(f"{counter} - {skill}")
```

```
0 - Html
1 - Css
2 - Js
3 - PHP
```

Sorted( )

sorted(iterable)

In [1]:

```
data=[4,2,15,8]
sorted(data)
```

Out[1]:

```
[2, 4, 8, 15]
```

In [3]:

```
sorted(data, reverse=True)
```

Out[3]:

```
[15, 8, 4, 2]
```

In [ ]:

```
# Lambda & Sorted
# Example 2: sorting data in dictionery based on a column
people=[{"name":"Jan", "age":39},
        {"name":"Fan", "age":25},
        {"name":"Dan", "age":34},
        {"name":"Kan", "age":27}]

sorted(people, key=lambda x: x["age"])
```

Reversed( )

reversed(iterable)

In [111…

```
myString = "Elzero"
list(reversed(myString))
```

Out[111…

```
['o', 'r', 'e', 'z', 'l', 'E']
```

In [112…

```
mySkills = ["Html", "Css", "Js", "PHP"]
list(reversed(mySkills))
```

Out[112…

```
['PHP', 'Js', 'Css', 'Html']
```

Round( )

Nearest digit

In [27]:

```
# round(number, numofdigits)
print(round(99.451))
print(round(99.501))
print(round(99.554, 2))
print(round(99.554, 3))
print(round(99.555, 2))
```

```
99
100
99.55
99.554
99.56
```

Abs( )

In [37]:

```
print(abs(100))
print(abs(-100))
print(abs(10.19))
print(abs(-10.19))
```

```
100
100
10.19
10.19
```

pow( )

Power

In [41]:

```
# pow(base, exp, mod) => Power
print(pow(2, 5))      # 2**5 = 2 * 2 * 2 * 2 * 2
print(pow(2, 5, 10))  # 2**5 = (2 * 2 * 2 * 2 * 2) % 10
pow(2, 5) /10
```

```
32
2
```

Out[41]:

```
3.2
```

print( )

In [31]:

```
# separetor
print("Hello Osama How Are You")
print("Hello", "Osama", "How", "Are", "You")

print("-"*50)

print("Hello @ Osama @ How @ Are @ You")
print("Hello", "Osama", "How", "Are", "You", sep=" @ ")  # sep: separetor
```

```
Hello Osama How Are You
Hello Osama How Are You
--------------------------------------------------
Hello @ Osama @ How @ Are @ You
Hello @ Osama @ How @ Are @ You
```

In [34]:

```
# End
print("First Line", end=" ")
print("Second Line")
print("Third Line")
```

```
First Line Second Line
Third Line
```

In [36]:

```
print("First Line", end=" %%%&&&&&&&& ")
print("Second Line")
print("Third Line")
```

```
First Line %%%&&&&&&&& Second Line
Third Line
```

In [33]:

```
print("First Line", end="\n")
print("Second Line")             #By default all the print function ends with "\n"
print("Third Line")
```

```
First Line
Second Line
Third Line
```

All( )

In [2]:

```
# All Elements Is True
x = [1, 2, 3, 4]
all(x)
```

Out[2]:

```
True
```

In [4]:

```
#Theres At Least One Element Is False
x = [1, 2, 3, 0]
all(x)[]
```

Out[4]:

```
False
```

In [5]:

```
x = [1, 2, 3, []]
all(x)
```

Out[5]:

```
False
```

In [6]:

```
x = [1, 2, 3, 4, []]

if all(x):

  print("All Elements Is True")

else:

  print("Theres At Least One Element Is False")
```

```
Theres At Least One Element Is False
```

Any( )

In [8]:

```
x = [3, 0, []]
any(x)
```

Out[8]:

```
True
```

In [9]:

```
x = [0, 0, []]
any(x)
```

Out[9]:

```
False
```

In [7]:

```
x = [0, 0, []]

if any(x):

  print("There's At Least One Element is True")

else:

  print("Theres No Any True Elements")
```

```
Theres No Any True Elements
```

Bin( )

Binery

In [10]:

```
bin(100)
```

Out[10]:

```
'0b1100100'
```

ID( )

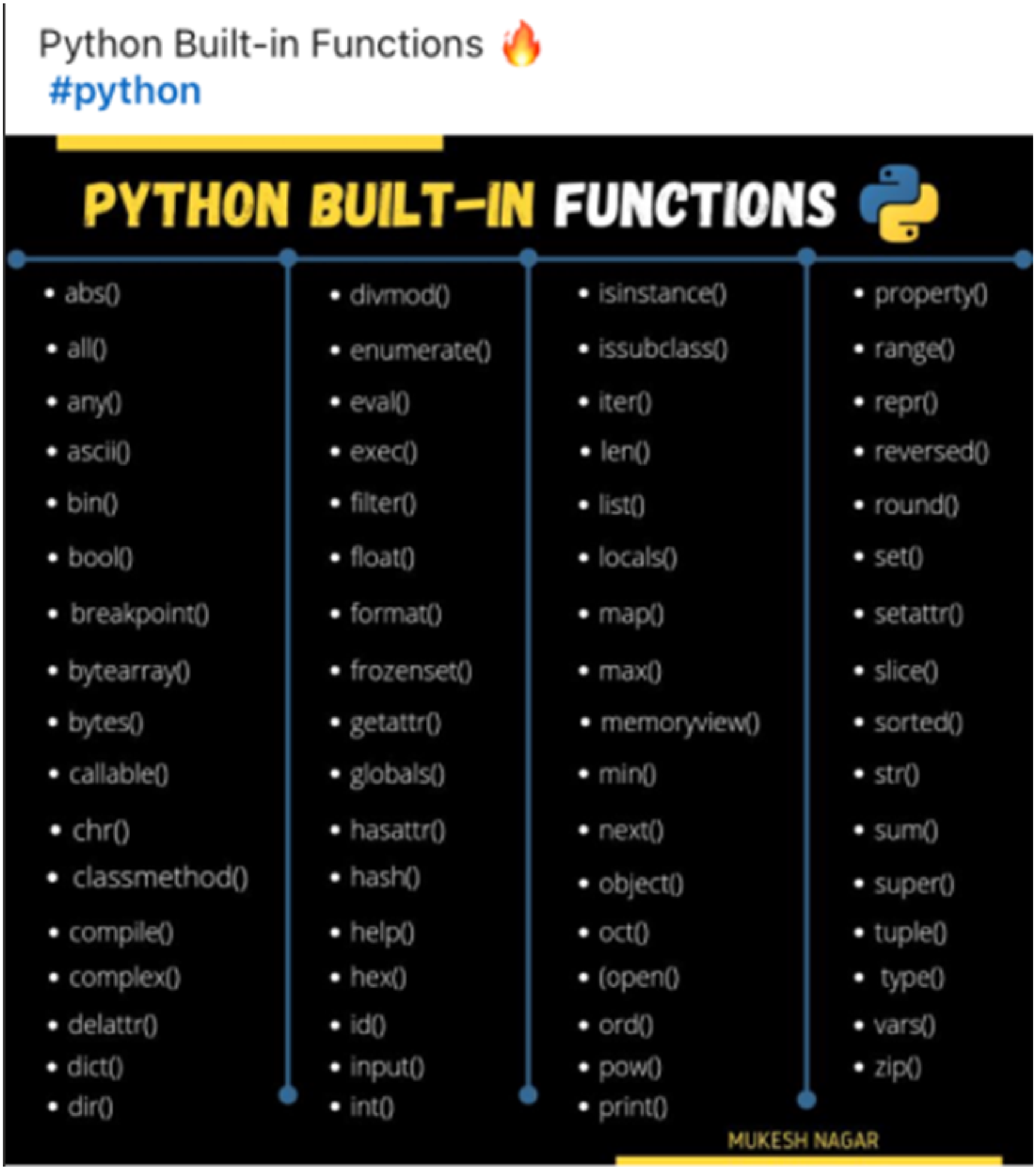
Memory ID

In [11]:

```
a = 1
b = 2

print(id(a))
print(id(b))
```

```
140729271985952
140729271985984
```

Built-in ( )



Python Built-in Functions 🔥
#python

PYTHON BUILT-IN FUNCTIONS

- abs()
- all()
- any()
- ascii()
- bin()
- bool()
- breakpoint()
- bytearray()
- bytes()
- callable()
- chr()
- classmethod()
- compile()
- complex()
- delattr()
- dict()
- dir()
- divmod()
- enumerate()
- eval()
- exec()
- filter()
- float()
- format()
- frozenset()
- getattr()
- globals()
- hasattr()
- hash()
- help()
- hex()
- id()
- input()
- int()
- isinstance()
- issubclass()
- iter()
- len()
- list()
- locals()
- map()
- max()
- memoryview()
- min()
- next()
- object()
- oct()
- (open()
- ord()
- pow()
- print()
- property()
- range()
- repr()
- reversed()
- round()
- set()
- setattr()
- slice()
- sorted()
- str()
- sum()
- super()
- tuple()
- type()
- vars()
- zip()

MUKESH NAGAR

String Formatting

Method 1: f-formatting

{} : place holder

In [109…

```
# Example 1
name = "Shola"
age = 35
position = "secretary"
```

```
New_string=f"The current {position} of ABC is a {age} year old guy named {name}"
New_string
```

Out[109…

```
'The current secretary of ABC is a 35 year old guy named Shola'
```

In [110…

```
print(f"The current {position} of ABC is a {age} year old guy named {name}")
```

```
The current secretary of ABC is a 35 year old guy named Shola
```

Method 2: %-formatting

%: place holder

In [111…

```
# Example 1
name = "Shola"
age = 35
position = "secretary"

New_string="The current %s of ABC is a %d year old guy named %s" %(position, age, name)
# s: string, d:digit, f:float

New_string
```

Out[111…

```
'The current secretary of ABC is a 35 year old guy named Shola'
```

In [112…

```
# Example 2
diameter = 3 # cm

New_string="The perimeter of a circle whose diameter is %.2f cm is %.3f cm" %(diameter,
# %.2f : 2 deciemel places

New_string
```

Out[112…

```
'The perimeter of a circle whose diameter is 3.00 cm is 9.429 cm'
```

Method 3: format function

In [113…

```
New_string="The current {:s} of ABC is a {:d} year old guy named {:s}".format(position,
# s: string, d:digit, f:float
New_string
```

Out[113…

```
'The current secretary of ABC is a 35 year old guy named Shola'
```

In [114…

```
New_string="The current {:s} of ABC is a {:f} year old guy named {:s}".format(position,
New_string
```

Out[114…

```
'The current secretary of ABC is a 35.000000 year old guy named Shola'
```

In [115…

```
# Rearrange items
New_string="The current {2} of CURTIN-EAGE is a {1} year old guy named {0}".format(posi
New_string
```

Out[115…

```
'The current Shola of CURTIN-EAGE is a 35 year old guy named secretary'
```

In [116…

```
New_string="The current {2:s} of CURTIN-EAGE is a {1:f} year old guy named {0:s}".forma
```

```
New_string
```

Out[116…

```
'The current Shola of CURTIN-EAGE is a 35.000000 year old guy named secretary'
```

In [117…

```
number = 10.14159
nwe_number = float(format(number, ".3f"))
nwe_number
```

Out[117…

```
10.142
```

In [ ]:

```

```

Truncate string

In [118…

```
myLongString='The current secretary of ABC is a 35 year old guy named Shola'
"Short string is %s" %myLongString
```

Out[118…

```
'Short string is The current secretary of ABC is a 35 year old guy named Shola'
```

In [119…

```
"Short string is %.15s" %myLongString
```

Out[119…

```
'Short string is The current sec'
```

In [120…

```
"Short string is {}" .format(myLongString)
```

Out[120…

```
'Short string is The current secretary of ABC is a 35 year old guy named Shola'
```

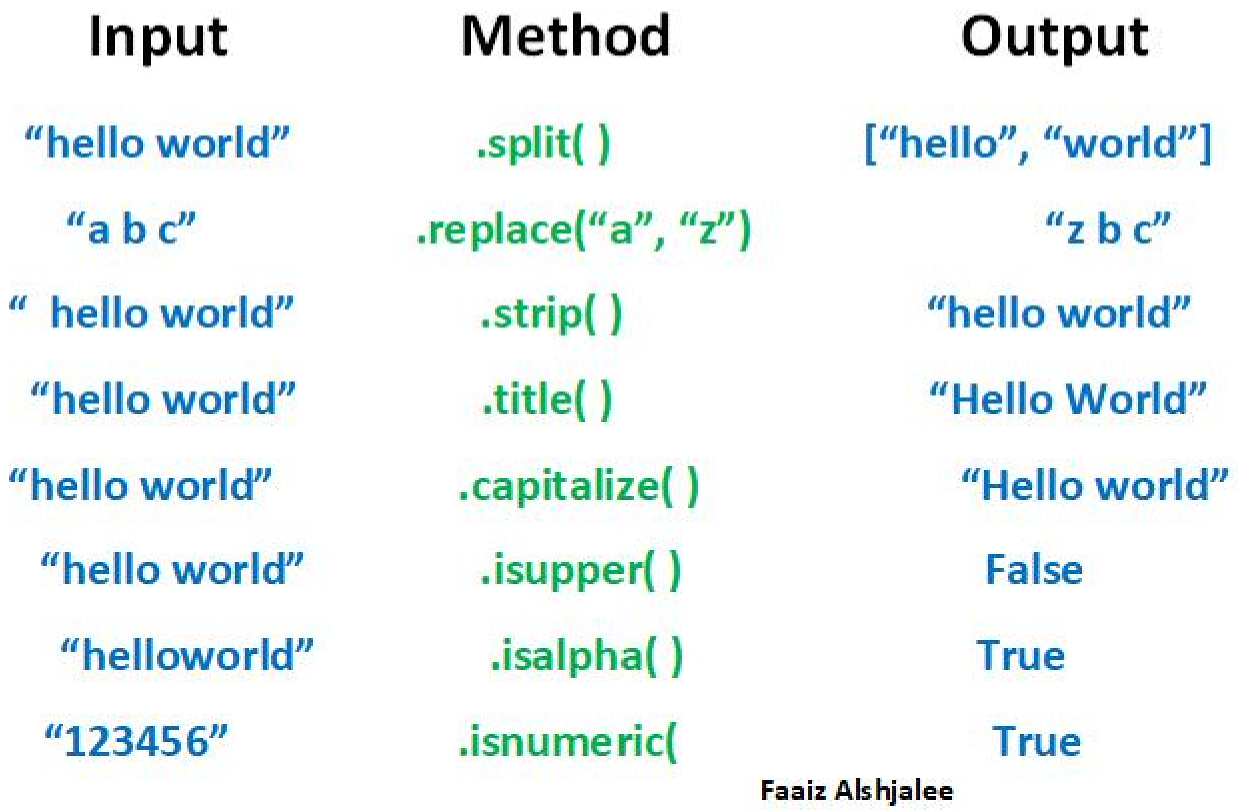
In [121…

```
"Short string is {:.12s}" .format(myLongString)
```

Out[121…

```
'Short string is The current '
```

String methods



Input	Method	Output
“hello world”	.split( )	[“hello”, “world”]
“a b c”	.replace(“a”, “z”)	“z b c”
“ hello world”	.strip( )	“hello world”
“hello world”	.title( )	“Hello World”
“hello world”	.capitalize( )	“Hello world”
“hello world”	.isupper( )	False
“helloworld”	.isalpha( )	True
“123456”	.isnumeric(	True

Faaiz Alshjalee

.split( ) method

The result is a list of the individual words making up newString

In [122…
```
# Examole 1
# Splitting a string
string = "hello world : how are you guys"
string.split(" ") # using space as my separator
```

Out[122…
```
['hello', 'world', ':', 'how', 'are', 'you', 'guys']
```

In [123…
```
input_file = 'account_ledger.txt'
file_name=input_file.split('.')
file_name
```

Out[123…
```
['account_ledger', 'txt']
```

In [124…
```
# Examole 2
# Splitting a file name
input_file = 'account_ledger.txt'
file_name=input_file.split('.')[0]  # using . as my separator
file_name
```

Out[124…
```
'account_ledger'
```

In [125…
```
file_extension=input_file.split('.')[1]
file_extension
```

Out[125…
```
'txt'
```

.split( ), max method

In [126…

```
# Examole 3
# Max split
string = "hello world : how are you guys"
string.split(" ", 3)
```

Out[126…

```
['hello', 'world', ':', 'how are you guys']
```

.rightsplit( ), max method

In [127…

```
# Right $ Max split
string = "hello world : how are you guys"
string.rsplit(" ", 2)
```

Out[127…

```
['hello world : how are', 'you', 'guys']
```

Concatenating strings: operation

In [128…

```
# let's combine both splitting and concatenating change the format of a file from .txt
input_file = 'account_ledger.txt'
output_file = input_file.split('.')[0] + '.csv'
output_file
```

Out[128…

```
'account_ledger.csv'
```

.replace( ) method

In [129…

```
# replacing an item in a string - variable_name.replace('old_item', 'new_item')
string = "hello world helow world"
string.replace("world", "global")
```

Out[129…

```
'hello global helow global'
```

In [130…

```
string
```

Out[130…

```
'hello world helow world'
```

Despite replacing "people" with "academics", string remains unchanged. Why?...Well, strings are immutable ordinarily. So, even though the change is valid, it doesn't affect the original string. To impose the change on the original string, we would have to reassign the change made to a new string as illiustrated below

In [131…

```
new_string = string.replace("world", "global")
new_string
```

Out[131…

```
'hello global helow global'
```

In [132…

```
new_string2 = string.replace("world", "global",1)
new_string2
```

```
'hello global helow world'
```

Out[132…

.join( ) method

In [133…

```
# Element in a list to string
myList = ["hello", "world", ":", "how", "are", "you", "gsys"]
" ".join(myList)
```

Out[133…

```
'hello world : how are you gsys'
```

In [134…

```
"-".join(myList)
```

Out[134…

```
'hello-world-:-how-are-you-gsys'
```

.strip( ) method

In [135…

```
# Example 1
"   Hello world  ".strip()
```

Out[135…

```
'Hello world'
```

In [136…

```
# Example 2
"####Hello world###".strip("#")
```

Out[136…

```
'Hello world'
```

In [137…

```
# Example 3
"@#@#Hello world@#@#".strip("@#")
```

Out[137…

```
'Hello world'
```

.rightstrip( ) method

In [138…

```
"   Hello world  ".rstrip()
```

Out[138…

```
'   Hello world'
```

.leftstrip( ) method

In [139…

```
"   Hello world  ".lstrip()
```

Out[139…

```
'Hello world  '
```

.title( ) method

In [140…

```
"hellow world".title()
```

Out[140…

```
'Hellow World'
```

In [141…

```
"hellow 4d world".title()
```

```
Out[141… 'Hellow 4D World'
```

.istitle( ) method

```
In [142… "hellow world".istitle()
```

```
Out[142… False
```

.capitalize( ) method

```
In [143… "hellow world".capitalize()
```

```
Out[143… 'Hellow world'
```

.upper( ) method

```
In [144… "hellow world".upper()
```

```
Out[144… 'HELLOW WORLD'
```

.isupper( ) method

```
In [145… "hellow world".isupper()
```

```
Out[145… False
```

.lower( ) method

```
In [146… "HELLOW WORLD".lower()
```

```
Out[146… 'hellow world'
```

.lower( ) method

```
In [147… "HELLOW WORLD".islower()
```

```
Out[147… False
```

"hellow world".islower()

.isalpha( ) method

```
In [148… "hellowworld".isalpha()
```

```
Out[148… True
```

```
In [149… "hellowworld3".isalpha()
```

```
Out[149… False
```

.isnumeric( ) method

In [150…
```
"123".isnumeric()
```

Out[150…
```
True
```

.isalnum( ) method

In [151…
```
"123".isalnum()
```

Out[151…
```
True
```

In [152…
```
"hellowworld123".isalnum()
```

Out[152…
```
True
```

.isspace() method

In [153…
```
" ".isspace()
```

Out[153…
```
True
```

.isidentifier() method

: is variable?

In [154…
```
"hellow_world".isidentifier()
```

Out[154…
```
True
```

In [155…
```
"hellow--world".isidentifier()
```

Out[155…
```
False
```

.zerofill( ) method

In [156…
```
a, b, c="1","10","100"
print (a)
print (b)
print (c)
```

```
1
10
100
```

In [157…
```
print (a.zfill(3))
print (b.zfill(3))
print (c.zfill(3))
```

```
001
010
100
```

.center( ) method

In [158…
```
"Python".center(0)
```
Out[158…
```
'Python'
```

In [159…
```
"Python".center(10)
```
Out[159…
```
'  Python  '
```

In [160…
```
"Python".center(10," ")
```
Out[160…
```
'  Python  '
```

In [161…
```
"Python".center(10,"#")
```
Out[161…
```
'##Python##'
```

.count( ) method

In [162…
```
"hellow world hellow world Hellow World".count("world")
```
Out[162…
```
2
```

In [163…
```
# count("string", start, end)
"hellow world hellow world Hellow World".count("world",0,14)
```
Out[163…
```
1
```

.swapcase( ) method

In [164…
```
"Python".swapcase()
```
Out[164…
```
'pYTHON'
```

.startswith( ) method

is starts with?

In [165…
```
"Python".startswith("P")
```
Out[165…
```
True
```

In [166…
```
"Python".startswith("y")
```
Out[166…
```
False
```

In [167…
```
# ("substring", start, end)
"Python".startswith("th",2,5)
```
Out[167…
```
True
```

.endswith( ) method

is ends with?

In [168…
```
"Python".endswith("P")
```

Out[168…
```
False
```

In [169…
```
"Python".endswith("n")
```

Out[169…
```
True
```

In [170…
```
"Python".endswith("t",0,3)
```

Out[170…
```
True
```

.index( ) method

In [171…
```
# index(SubString, start, end)
"hellow world".index("w")
```

Out[171…
```
5
```

In [172…
```
"hellow world".index("ow")
```

Out[172…
```
4
```

In [173…
```
# "hellow world".index("w",0,5) # this results in error
```

.find( ) method

In [174…
```
"hellow world".find("w",0,6)
```

Out[174…
```
5
```

In [175…
```
"hellow world".find("w",0,5) # results in -1 instead of error
```

Out[175…
```
-1
```

.just( ) method

Justify

.right just( ) method

In [176…
```
"Python".rjust(10)
```

Out[176…
```
'    Python'
```

In [177…
```
"Python".rjust(10,"#")
```

Out[177…

```
'####Python'
```

.left just( ) method

In [178…

```
"Python".ljust(10,"#")
```

Out[178…

```
'Python####'
```

.splitlines( ) method

In [179…

```
a=""" First line
Second line
Third line"""
a
```

Out[179…

```
' First line\nSecond line\nThird line'
```

In [180…

```
a.splitlines()
```

Out[180…

```
[' First line', 'Second line', 'Third line']
```

In [181…

```
b=' First line\nSecond line\nThird line'
b.splitlines()
```

Out[181…

```
[' First line', 'Second line', 'Third line']
```

.expandtabs( ) method

In [182…

```
b=' First line\tSecond line\tThird line'
b.expandtabs(30)
```

Out[182…

```
' First line                   Second line                   Third line'
```

More

Object multiplication

In [44]:

```
x="Hello "*3
x
```

Out[44]:

```
'Hello Hello Hello '
```

In [45]:

```
x=[1,2,3]*3
x
```

Out[45]:

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

In [41]:

```
x=[[1,2,3]]*3
x
```

```
Out[41]: [[1, 2, 3], [1, 2, 3], [1, 2, 3]]
```

```
In [42]: x=(1,2,3)*3
         x
```

```
Out[42]: (1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Swap values

```
In [50]: width,higth=100,500
         width,higth
```

```
Out[50]: (100, 500)
```

```
In [51]: width,higth=higth,width
         width,higth
```

```
Out[51]: (500, 100)
```

```
In [52]: width,higth,z=higth,width,5
         width,higth,z
```

```
Out[52]: (100, 500, 5)
```

4 Merge Dictionaries: {**d1, **d2}

```
In [53]: d1={"name":"Jah", "age":33}
         d2={"name":"Nah", "sallery":5000}
         d={**d1,**d2}
         d
```

```
Out[53]: {'name': 'Nah', 'age': 33, 'sallery': 5000}
```

```
In [ ]: 
```