23-06-2025

In [ ]:
```
Interpreter :Interpreter translates and runs the code line by line, showing erro
Compiler    :Compiler translates the entire code at once into machine code befor
        ✅ Interpreter is slower but easier for debugging.
        ✅ Compiler is faster but shows all errors after compilation.
```

In [ ]:
```
Note: Python is both Interpreter and compiler Language
Note: .ipynb -->Interactive Python Notebook
```

In [ ]:
```
The name "Python" came from the British comedy TV show "Monty Python's Flying Ci
```

In [ ]:
```

```

Data Structures

In [ ]:
```
                        +-----------------------------+
                        |     Python Data Structures  |
                        +---------------+-------------+
                                        |
       +--------------------------------+---------------------------------+
       |                                                                  |
+------------------------+                                  +---------------------
|   Built-in Data Types  |                                  |   User-defined Data
+------------------------+                                  +---------------------
       |                                                                  |
+------+-----+-----+-----+                                     +-----+-----+-----+
|  List     Tuple  Set  Dictionary                            | Stack Queue Deque
+-------------------------------------------------------------+-----------------
|  Array (via list)                                         | Linked List (Sing
                                                            | Hash Table
                                                            | Heap
                                                            +-----------------
```

In [ ]:
```
✅ Data Type: It defines the type of data a variable can hold (like int, float,
   --> User can Assign only 1 value to the Variable
✅ Data Structure: It defines how data is stored and organized (like list, stacl
   --> User can Assign multiple values to the variable

📌 In short: Data type is about value, data structure is about organization.
```

List

In [ ]:
```

```

In [1]:
```
l = []
l
```

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

In [3]:
```
type(l)
```

Out[3]: list

In [4]:
```
print(type(l))
```

```
<class 'list'>
```

In [5]:
```
l.append(10)
l       #-----------> append() adds the value/data at last/end of the List
```

Out[5]: [10]

In [6]:
```
l.append(20)
l.append(30)
l.append(40)
l.append(50)
l
```

Out[6]: [10, 20, 30, 40, 50]

In [7]:
```
l.append(10,20)
l       ------------------> append() takes exactly one argument (2 given)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 l.append(10,20)
      2 l

TypeError: list.append() takes exactly one argument (2 given)
```

In [9]:
```
l.append(10)
l          #-------------> Duplicate values are Allowed
```

Out[9]: [10, 20, 30, 40, 50, 10, 10]

In [ ]:
```

```

24-06-2025

In [2]:
```
l = [10, 20, 30, 40, 50, 10]
print(l)
```

```
[10, 20, 30, 40, 50, 10]
```

In [3]:
```
l[0]
```

Out[3]: 10

In [4]:
```
l[2]
```

Out[4]: 30

In [5]:
```
l[-1]
```

Out[5]: 10

In [7]:
```
l1 = l.copy()
l1
```

Out[7]: [10, 20, 30, 40, 50, 10]

In [8]:
```
l==l1
```

Out[8]: True

In [9]:
```
id(l) == id(l1)
```

Out[9]: False

In [10]:
```
l.count(10)
```

Out[10]: 2

In [13]:
```
l.count(40)
```

Out[13]: 1

In [15]:
```
l1.clear()
l1
```

Out[15]: []

In [20]:
```
l
```

Out[20]: [10, 20, 30, 40, 50, 10]

In [25]:
```
del l[0]
l
```

Out[25]: [30, 40, 50, 10]

In [26]:
```
l
```

Out[26]: [30, 40, 50, 10]

In [28]:
```
l2 = []
l2
```

Out[28]: []

In [30]:
```
l2.append(9)
l2.append('nit')
l2.append(True)
l2.append(1+2j)
l2.append(10.5)
l2
```

Out[30]: [9, 'nit', True, (1+2j), 10.5]

In [31]:
```
print(l)
print(l2)
```

```
[30, 40, 50, 10]
[9, 'nit', True, (1+2j), 10.5]
```

In [32]:
```
l.index(50)
```

Out[32]: 2

In [34]:
```
l2.index(True)
```

Out[34]: 2

In [35]:
```
l2[1][1]    #Nested Indexing
```

Out[35]: 'i'

In [36]:
```
l[:]          # List Slicing
```

Out[36]: [30, 40, 50, 10]

In [37]:
```
l[:3]
```

Out[37]: [30, 40, 50]

In [40]:
```
l[3:]
```

Out[40]: [10]

In [41]:
```
l[4:]
```

Out[41]: []

In [42]:
```
l2
```

Out[42]: [9, 'nit', True, (1+2j), 10.5]

In [44]:
```
l2[0:5:2]
```

Out[44]: [9, True, 10.5]

In [45]:
```
l
```

Out[45]: [30, 40, 50, 10]

In [48]:
```
l.insert(3,70)
l
```

Out[48]: [30, 40, 50, 70, 60, 60, 10]

In [49]:
```
l.pop(4)
```

Out[49]: 60

In [50]:
```
l
```

Out[50]: [30, 40, 50, 70, 60, 10]

In [51]:
```
l.insert(5,80)
l
```

Out[51]: [30, 40, 50, 70, 60, 80, 10]

In [52]:
```
l.pop()
```

Out[52]: 10

In [53]:
```
l
```

Out[53]: [30, 40, 50, 70, 60, 80]

25-06-2025

List Membership

In [3]:
```
l = [100,10,15,20,30,40]
l
```

Out[3]: [100, 10, 15, 20, 30, 40]

In [6]:
```
100 in l
```

Out[6]: True

In [7]:
```
1000 in l
```

Out[7]: False

In [8]:
```
for i in l:
    print(i)
```

```
100
10
15
20
30
40
```

In [9]:
```
l
```

Out[9]: [100, 10, 15, 20, 30, 40]

In [10]:
```
for i in enumerate(l):
    print(i)
```

```
(0, 100)
(1, 10)
(2, 15)
(3, 20)
(4, 30)
(5, 40)
```

In [12]:
```
l2 = [9,True,9.3]
l2
```

Out[12]:
```
[9, True, 9.3]
```

In [13]:
```
print(l)
print(l2)
```

```
[100, 10, 15, 20, 30, 40]
[9, True, 9.3]
```

In [15]:
```
l2.extend(l)
l2
```

Out[15]:
```
[9, True, 9.3, 100, 10, 15, 20, 30, 40, 100, 10, 15, 20, 30, 40]
```

In [16]:
```
l
```

Out[16]:
```
[100, 10, 15, 20, 30, 40]
```

In [17]:
```
l.sort()
```

In [18]:
```
l
```

Out[18]:
```
[10, 15, 20, 30, 40, 100]
```

In [20]:
```
l.sort(reverse=True)
l
```

Out[20]:
```
[100, 40, 30, 20, 15, 10]
```

In [21]:
```
l2
```

Out[21]:
```
[9, True, 9.3, 100, 10, 15, 20, 30, 40, 100, 10, 15, 20, 30, 40]
```

In [23]:
```
l3 = [1,2.3,'nit',True]
l3
```

Out[23]:
```
[1, 2.3, 'nit', True]
```

In [26]:
```
l3.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 1
----> 1 l3.sort()

TypeError: '<' not supported between instances of 'str' and 'float'
```

In [27]:
```
l4 = ['a','z','n','m']
l4
```

Out[27]: ['a', 'z', 'n', 'm']

In [30]:
```
l4.sort()
l4
```

Out[30]: ['a', 'm', 'n', 'z']

In [33]:
```
l4.reverse()
l4
```

Out[33]: ['z', 'n', 'm', 'a']

All & Any

In [34]:
```
l
```

Out[34]: [100, 40, 30, 20, 15, 10]

In [35]:
```
all(l)
```

Out[35]: True

In [36]:
```
any(l)
```

Out[36]: True

TUPLE

In [38]:
```
t = ()
t
```

Out[38]: ()

In [39]:
```
type(t)
```

Out[39]: tuple

In [40]:
```
print(type(t))
```

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

In [41]:
```
t1 = (10,20,30)
t1
```

Out[41]: (10, 20, 30)

In [43]:
```
sbi = ('alexa',6448651648,'jhbhjsd75654dgv',268456788)
sbi
```

Out[43]: ('alexa', 6448651648, 'jhbhjsd75654dgv', 268456788)

In [44]:
```
sbi[1] = 20  # Tuple is Immutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 sbi[1] = 20

TypeError: 'tuple' object does not support item assignment
```

In [45]:
```
t1.count(10)
```

Out[45]: 1

In [46]:
```
t1.index(20)
```

Out[46]: 1

In [47]:
```
t1
```

Out[47]: (10, 20, 30)

In [48]:
```
for i in t1:
    print(i)
```

```
10
20
30
```

In [49]:
```
for i in enumerate(t1):
    print(i)
```

```
(0, 10)
(1, 20)
(2, 30)
```

In [50]:
```
t2 = t1*3
t2
```

Out[50]: (10, 20, 30, 10, 20, 30, 10, 20, 30)

In [52]:
```
t2[:6]
```

Out[52]: (10, 20, 30, 10, 20, 30)

In [53]:
```
t2[2:6]
```

Out[53]: (30, 10, 20, 30)

In [63]:
```
t1 = (10, 20, 30)
t1
```

Out[63]: (10, 20, 30)

In [64]:
```
del t1
```

In [65]:
```
t1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[65], line 1
----> 1 t1

NameError: name 't1' is not defined
```

In [ ]:

In [ ]: