```
In [1]: print("TejaswiSaiKumar")

        TejaswiSaiKumar

In [2]: a = 1
        a

Out[2]: 1

In [3]: type(a)

Out[3]: int

In [4]: b = 10.23
        type(b)

Out[4]: float

In [5]: c = 'Teja'
        print(c)
        type(c)

        Teja
Out[5]: str

In [6]: d = True
        type(d)

Out[6]: bool
```

Boolean variables - True and False

```
In [7]: True - False

Out[7]: 1

In [8]: True * False

Out[8]: 0

In [9]: True / False

        ---------------------------------------------------------------------------
        ZeroDivisionError                         Traceback (most recent call last)
        Cell In[9], line 1
        ----> 1 True / False

        ZeroDivisionError: division by zero

In [10]: e = 13+15j
         type(e)

Out[10]: complex

In [11]: e.imag

Out[11]: 15.0
```

```
In [12]: """ Printing the real number
         from a complex number"""
         e.real

Out[12]: 13.0

In [13]: _a = 15
         _a

Out[13]: 15

In [14]: #Typecasting
         str(_a)+c

Out[14]: '15Teja'

In [15]: i = input()

         15

In [16]: i

Out[16]: '15'

In [17]: type(i)

Out[17]: str

In [18]: j = int(input())

         15

In [19]: print(j)
         type(j)

         15
Out[19]: int

In [20]: t = "teja"
         print(t[-1])
         print(t[3])

         a
         a

In [21]: List = [13, 15, "Sukesh", "Teja", 'Sukesh and Teja']
         print(List)
         print(type(List))
         print(List[3])
         print(List[-2])

         [13, 15, 'Sukesh', 'Teja', 'Sukesh and Teja']
         <class 'list'>
         Teja
         Teja

In [22]: """immutable and mutable
         Lists are mutable
         Strings are immutabe"""

         List[4] = 'KrishnaMohan and Padmaja'   #Lists are mutable
         print(List)
```

```
[13, 15, 'Sukesh', 'Teja', 'KrishnaMohan and Padmaja']
```

```
In [23]: t[3] = 'n'
         print(t)     #Strings are immutable
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[23], line 1
----> 1 t[3] = 'n'
      2 print(t)

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

Operators

```
In [24]: 13+15

Out[24]: 28

In [25]: 15-13

Out[25]: 2

In [26]: 15*13

Out[26]: 195

In [27]: 15/13

Out[27]: 1.1538461538461537

In [28]: 15%13

Out[28]: 2

In [29]: #(15 to the power 13)
         15**13

Out[29]: 1946195068359375

In [30]: 24//6

Out[30]: 4

In [31]: 1>2

Out[31]: False

In [32]: 2<3

Out[32]: True

In [33]: # Comparison operator
         2 == 2

Out[33]: True
```

In [34]:
```
13>=15
```
Out[34]:
```
False
```
In [35]:
```
15>=14
```
Out[35]:
```
True
```
In [36]:
```
# Logical operator - AND, OR, NOT
print(True and True)
print(True and False)
print(False and False)
print(False and True)
```
```
True
False
False
False
```
In [37]:
```
print(True or True)
print(True or False)
print(False or False)
print(False or True)
```
```
True
True
False
True
```
In [38]:
```
print(not True)
print(not False)
```
```
False
True
```
In [39]:
```
"""Bitwise operator - Convert the dataset into a bitwise
Symbols: OR - |, AND - &"""

print(13 | 15)
print(bin(13))
print(bin(15))
```
```
15
0b1101
0b1111
```
In [40]:
```
~13
```
Out[40]:
```
-14
```
In [41]:
```
#Right shif operator
print(30 >> 2) #You will loose last 2 binary digits

#Left shift operator
print(35 << 3)    # You will gain three 0's at the last
```
```
7
280
```
In [42]:
```
t = 15
t
```
Out[42]:
```
15
```

In [43]:
```
t += 5
t
```
Out[43]:
```
20
```

Conditions

In [44]:
```
t = int(input("Enter the value of t:"))

if t > 15:
    print("If block executed")
elif t <= 15:
    print("elif block executed")
else:
    print("else block executed")
```
```
Enter the value of t:25
If block executed
```

In [45]:
```
#Loop
t = 16
s = 13

while s < t:
    s += 1
    if s == 20:
        break
    print(s)
else:
    print("else executed when while condition fails")
```
```
14
15
16
else executed when while condition fails
```

In [46]:
```
#Loop
t = 16
s = 13

while s < t:
    s += 1
    if s == 14:
        continue
    print(s)
else:
    print("else executed when while condition fails")
```
```
15
16
else executed when while condition fails
```

In [47]:
```
t = "teja"

for i in t:
    print(i)
```
```
t
e
j
a
```

In [48]:
```
List = [13, 15, "Sukesh", "Teja", 'Sukesh and Teja']
for j in List:

    if j == 'Teja':
        break
    print(j)
else:
    print("else will be executed when condition fails")
```

```
13
15
Sukesh
```

In [49]:
```
List = [13, 15, "Sukesh", "Teja", 'Sukesh and Teja']
for j in List:

    if j == 'Teja':
        continue
    print(j)
else:
    print("else will be executed when condition fails")
```

```
13
15
Sukesh
Sukesh and Teja
else will be executed when condition fails
```

In [50]:
```
range(10)
```

Out[50]:
```
range(0, 10)
```

In [51]:
```
list(range(10))
```

Out[51]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Tuple

In [52]:
```
"""
1. The difference b/w List and Tuple is paranthesis
List - []
Tuple - ()
2. Two in-built functions are available in a tuple - count() and index().
3. Tuple are immutable whereas Lists are mutable.
"""

Tuple = ()
type(Tuple)
```

Out[52]:
```
tuple
```

In [53]:
```
t1 = (13, 15, "Sukesh", "Teja", 'Sukesh and Teja', 13+15j, True, 1)
print(t1)
print(type(t1))
len(t1)

# Check wheteher an element is available in a tuple or not ?
print("Sukesh and Teja"in t1)
print(2 in t1)
```

```
(13, 15, 'Sukesh', 'Teja', 'Sukesh and Teja', (13+15j), True, 1)
<class 'tuple'>
True
False
```

In [54]:
```
#Extract information from tuple
t1[5]
```

Out[54]:
```
(13+15j)
```

In [55]:
```
#Reverse of a tuple
t1[::-1]
```

Out[55]:
```
(1, True, (13+15j), 'Sukesh and Teja', 'Teja', 'Sukesh', 15, 13)
```

In [56]:
```
t1
```

Out[56]:
```
(13, 15, 'Sukesh', 'Teja', 'Sukesh and Teja', (13+15j), True, 1)
```

In [57]:
```
#Extract required info from a tuple
t1[0:4]
```

Out[57]:
```
(13, 15, 'Sukesh', 'Teja')
```

In [58]:
```
# Internally system stores True as 1. In that case, we have two 1's in our tuple.
print(t1.count(True))
print(t1.count(1))
print(t1.count(13))
```

```
2
2
1
```

In [59]:
```
print(t1.index(13))
print(t1.index("Sukesh and Teja"))
```

```
0
4
```

In [60]:
```
List = [1, 20, 13, 15]
List
```

Out[60]:
```
[1, 20, 13, 15]
```

In [61]:
```
#Tuples are immutable
t1[5] = 'KrishnaMohan'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[61], line 2
      1 #Tuples are immutable
----> 2 t1[5] = 'KrishnaMohan'

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

In [62]:
```
for t in t1:
    print(t, type(t))
```

```
13 <class 'int'>
15 <class 'int'>
Sukesh <class 'str'>
Teja <class 'str'>
Sukesh and Teja <class 'str'>
(13+15j) <class 'complex'>
True <class 'bool'>
1 <class 'int'>
```

In [63]:
```
t1 * 2
```

Out[63]:
```
(13,
 15,
 'Sukesh',
 'Teja',
 'Sukesh and Teja',
 (13+15j),
 True,
 1,
 13,
 15,
 'Sukesh',
 'Teja',
 'Sukesh and Teja',
 (13+15j),
 True,
 1)
```

In [64]:
```
#Replicating the tuple
t2 = (1, 20, 13, 15)
t2 * 3
```

Out[64]:
```
(1, 20, 13, 15, 1, 20, 13, 15, 1, 20, 13, 15)
```

In [65]:
```
max(t1)
```

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

TypeError: '>' not supported between instances of 'str' and 'int'
```

In [66]:
```
max(t2)
```

Out[66]:
```
20
```

In [67]:
```
min(t2)
```

Out[67]:
```
1
```

In [68]:
```
t1 = (1, 2, 3, 4)
t2 = (5, 6, 7, 8)
L1 = [9, 10, 11, 12]
t3 = (t1, t2, L1)
print(t3)
```

```
((1, 2, 3, 4), (5, 6, 7, 8), [9, 10, 11, 12])
```

In [69]:
```
del t3
```

In [70]:
```
t3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[70], line 1
----> 1 t3

NameError: name 't3' is not defined
```

Dictionary

In [71]:
```
"""
Dictioanry starts with the following paranthesis -  {}.
1. Set of KEY and VALUE seperated by a ":".
2. KEY can be a numeric format (int, float), boolean (or) a string represented with
3. The format of a key in a dictionary can be a TUPLE but it cannot accept as a LIS
4. VALUE can be a List, Tuple, Set, dictionary(Nested dictioanry).
5. We can update the dictionary (add, delete).
6. KEY behaves as an 'index' in the dictionary.
7. Dictionaries inside a TUPLE is possible.
"""

d = {}
type(d)
```

Out[71]:
```
dict
```

In [72]:
```
d1 = {"name": "TejaswiSaiKumar", "email": "atoz1tounlimited", "domain" : "@gmail.cc
print(d1)
print(type(d1))
```

```
{'name': 'TejaswiSaiKumar', 'email': 'atoz1tounlimited', 'domain': '@gmail.com',
'mobile_number': 9876543210}
<class 'dict'>
```

In [73]:
```
d2 = {"name": "TejaswiSaiKumar", "name": "Teja"}
print(d2)
```

```
{'name': 'Teja'}
```

In [74]:
```
d3 = {6341703 : "TejaswiSaiKumar"}
print(d3)
```

```
{6341703: 'TejaswiSaiKumar'}
```

In [75]:
```
d4 = {6341.703 : "Teja"}
print(d4)
```

```
{6341.703: 'Teja'}
```

In [76]:
```
d5 = {True : "TSK"}
print(d5)
```

```
{True: 'TSK'}
```

In [77]:
```
d6 = {(13,15): "Sukesh and Teja"}
print(d6)
```

```
{(13, 15): 'Sukesh and Teja'}
```

In [78]:
```
d7 = {[13, 15]: "Sukesh and Teja"}
print(d7)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[78], line 1
----> 1 d7 = {[13, 15]: "Sukesh and Teja"}
      2 print(d7)

TypeError: unhashable type: 'list'
```

In [79]:
```
d8 = {{13, 15}: "Sukesh and Teja"}
print(d8)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[79], line 1
----> 1 d8 = {{13, 15}: "Sukesh and Teja"}
      2 print(d8)

TypeError: unhashable type: 'set'
```

In [80]:
```
d9 = {{"Sukesh" : 13} : "KrishnaMohan"}
print(d9)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[80], line 1
----> 1 d9 = {{"Sukesh" : 13} : "KrishnaMohan"}
      2 print(d9)

TypeError: unhashable type: 'dict'
```

In [81]:
```
d10 = {"course_name": ["Machine Learning", "Deep Learning", "Geneative AI"]}
print(d10)
```

```
{'course_name': ['Machine Learning', 'Deep Learning', 'Geneative AI']}
```

In [82]:
```
d11 = {"key" : (1, 20, 13, 15)}
print(d11)
```

```
{'key': (1, 20, 13, 15)}
```

In [85]:
```
d12 = {
    "key" : {1, 20, 13, 15}
}
print(d12)
```

```
{'key': {1, 20, 13, 15}}
```

In [86]:
```
#Dictioanry inside a Dictionary = Nested Dictionary
d13 = {"key" : {"Teja" : 15, "Sukesh": 16}}
print(d13)
```

```
{'key': {'Teja': 15, 'Sukesh': 16}}
```

In [87]:
```
d14 = {
    "course_names" : ["Machine Learning", "Deep Learning", "Generative AI"],
    "start_date": (8, 1, 8),
    "professor_name": {"Sukesh", "Teja", "Krishna"}
}
d14
```

Out[87]:
```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'],
 'start_date': (8, 1, 8),
 'professor_name': {'Krishna', 'Sukesh', 'Teja'}}
```

In [88]:
```
d14["time"] = (14, 16, 8)
print(d14)
```
```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8)}
```
In [89]:
```
d14["start_date"]
```
Out[89]:
```
(8, 1, 8)
```
In [90]:
```
print(d14["professor_name"])
print(type(d14["professor_name"]))
```
```
{'Teja', 'Krishna', 'Sukesh'}
<class 'set'>
```
In [91]:
```
d14["student_name"] = "TejaswiSaiKumar"
print(d14)
```
```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8), 'student_name': 'TejaswiSaiKumar'}
```
In [92]:
```
d14["student_name"].upper()
```
Out[92]:
```
'TEJASWISAIKUMAR'
```
In [93]:
```
print(d13)
print(d13["key"])
print(type(d13["key"]))

#Printig the value inside a Nested dictioanry
print(d13["key"]["Teja"])

#Add info to a dictionary d13
d13["Head"] = "Krishna"
print(d13)
```
```
{'key': {'Teja': 15, 'Sukesh': 16}}
{'Teja': 15, 'Sukesh': 16}
<class 'dict'>
15
{'key': {'Teja': 15, 'Sukesh': 16}, 'Head': 'Krishna'}
```
In [94]:
```
del d13["Head"]
```
In [95]:
```
print(d13)
```
```
{'key': {'Teja': 15, 'Sukesh': 16}}
```
In [96]:
```
# in-built functions of a dictionary
print(d10)
```
```
{'course_name': ['Machine Learning', 'Deep Learning', 'Geneative AI']}
```
In [97]:
```
d10.clear()
```
In [98]:
```
d10
```
Out[98]:
```
{}
```

In [99]:
```
#Check the total number of key:value pairs in a dictionary
print(d13)
print(len(d13))

print(d14)
print(len(d14))
```

```
{'key': {'Teja': 15, 'Sukesh': 16}}
1
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8), 'student_name': 'TejaswiSaiKumar'}
5
```

In [100…
```
#Extract all the keys available in a dictionary - keys()
print(d14)
d14.keys()
```

```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8), 'student_name': 'TejaswiSaiKumar'}
```
Out[100]:
```
dict_keys(['course_names', 'start_date', 'professor_name', 'time', 'student_nam
e'])
```

In [101…
```
# Extract all the values available in a dictionary - values()
print(d14)
d14.values()
```

```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8), 'student_name': 'TejaswiSaiKumar'}
```
Out[101]:
```
dict_values([['Machine Learning', 'Deep Learning', 'Generative AI'], (8, 1, 8),
{'Teja', 'Krishna', 'Sukesh'}, (14, 16, 8), 'TejaswiSaiKumar'])
```

In [102…
```
# Convert into a proper list
print(list(d14.keys()))
print(list(d14.values()))
```

```
['course_names', 'start_date', 'professor_name', 'time', 'student_name']
[['Machine Learning', 'Deep Learning', 'Generative AI'], (8, 1, 8), {'Teja', 'Kris
hna', 'Sukesh'}, (14, 16, 8), 'TejaswiSaiKumar']
```

From the above output, we can say that - In a list, we have a list, tuple, set, string

In [103…
```
# Extract a list of key-value pairs - items()
list(d14.items())
```

Out[103]:
```
[('course_names', ['Machine Learning', 'Deep Learning', 'Generative AI']),
 ('start_date', (8, 1, 8)),
 ('professor_name', {'Krishna', 'Sukesh', 'Teja'}),
 ('time', (14, 16, 8)),
 ('student_name', 'TejaswiSaiKumar')]
```

From the above output, e can say that - We can see a 5 pair of key-value elements

In [104…
```
# copy() in a dictionary
# creates the data in a new space again.
#
d15 = d14.copy()
```

```
print(d14)
print(d15)
```

```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8), 'student_name': 'TejaswiSaiKumar'}
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8), 'student_name': 'TejaswiSaiKumar'}
```

In [105…
```
del d14["student_name"]
print(d14)
print(d15)
```

```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8)}
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8), 'student_name': 'TejaswiSaiKumar'}
```

In [106…
```
d16 = d14
print(d16)
```

```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8)}
```

In [107…
```
#pop()
print(d14)
d14.pop("time")
```

```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'], 'start_da
te': (8, 1, 8), 'professor_name': {'Teja', 'Krishna', 'Sukesh'}, 'time': (14, 16,
8)}
```

Out[107]:
```
(14, 16, 8)
```

In [108…
```
d14
```

Out[108]:
```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'],
 'start_date': (8, 1, 8),
 'professor_name': {'Krishna', 'Sukesh', 'Teja'}}
```

In [109…
```
d14.pop("start_date")
```

Out[109]:
```
(8, 1, 8)
```

In [110…
```
d14
```

Out[110]:
```
{'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'],
 'professor_name': {'Krishna', 'Sukesh', 'Teja'}}
```

In [111…
```
#fromkeys() - returns a dictionary with the specified keys and the specified values
d.fromkeys(("PKM", "PBSP"), ("Sukesh", "Teja"))
```

Out[111]:
```
{'PKM': ('Sukesh', 'Teja'), 'PBSP': ('Sukesh', 'Teja')}
```

In [112…
```
d17 = {
    "key1" : "value1",
    "key2" : "value2",
    "key3" : "value3"
}
print(d17)
```

```
d18 = {
    "key4" : "value4",
    "key5" : "value5",
    "key6" : "value6"
}
print(d18)
```

```
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
{'key4': 'value4', 'key5': 'value5', 'key6': 'value6'}
```

In [113…

```
(d17, d18)
```

Out[113]:

```
({'key1': 'value1', 'key2': 'value2', 'key3': 'value3'},
 {'key4': 'value4', 'key5': 'value5', 'key6': 'value6'})
```

In [114…

```
#update()
d17.update(d18)
print(d17)
print(d18)
```

```
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3', 'key4': 'value4', 'key5':
'value5', 'key6': 'value6'}
{'key4': 'value4', 'key5': 'value5', 'key6': 'value6'}
```

In [115…

```
print(d18)
d18.update(d17)
print(d18)
print(d17)
```

```
{'key4': 'value4', 'key5': 'value5', 'key6': 'value6'}
{'key4': 'value4', 'key5': 'value5', 'key6': 'value6', 'key1': 'value1', 'key2':
'value2', 'key3': 'value3'}
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3', 'key4': 'value4', 'key5':
'value5', 'key6': 'value6'}
```

In [116…

```
#get()

#get() in dictioanry never give us an error
print(d17.get("PKM"))

print(d17["key1"])
print(d17.get("key1"))
```

```
None
value1
value1
```

Dictionary comprehensions

In [117…

```
"""I want to print the square of keys as values in my dictionary"""
{s : s**2 for s in range(1,11)}
```

Out[117]:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

In [118…

```
"""I want to print my values that are 5 times more than its respective key in my di
{s : s + 5 for s in range(1,11)}
```

Out[118]:

```
{1: 6, 2: 7, 3: 8, 4: 9, 5: 10, 6: 11, 7: 12, 8: 13, 9: 14, 10: 15}
```

In [119…

```
"""I want to print my values that are logarithemic of its respective key in my dict
import math
d19 = {s : math.log2(s) for s in range(1,11)}
d19
```

```
Out[119]: {1: 0.0,
           2: 1.0,
           3: 1.584962500721156,
           4: 2.0,
           5: 2.321928094887362,
           6: 2.584962500721156,
           7: 2.807354922057604,
           8: 3.0,
           9: 3.169925001442312,
           10: 3.321928094887362}
```

```
In [120…  d14
```

```
Out[120]: {'course_names': ['Machine Learning', 'Deep Learning', 'Generative AI'],
           'professor_name': {'Krishna', 'Sukesh', 'Teja'}}
```

```
In [121…  #Check whetener 'course_names' is availabe or not in a dictionary
          'course_names' in d14
```

```
Out[121]: True
```

```
In [122…  d19
```

```
Out[122]: {1: 0.0,
           2: 1.0,
           3: 1.584962500721156,
           4: 2.0,
           5: 2.321928094887362,
           6: 2.584962500721156,
           7: 2.807354922057604,
           8: 3.0,
           9: 3.169925001442312,
           10: 3.321928094887362}
```

```
In [123…  d19.keys()
```

```
Out[123]: dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
In [124…  # Extract the values of a even key from a dictionary.
          for e in d19.keys():
              if e%2 == 0:
                  print(e, d19[e])
```

```
2 1.0
4 2.0
6 2.584962500721156
8 3.0
10 3.321928094887362
```

```
In [ ]:
```