```
✍Words are called 'Tokens' in python For example:---
✍"My name is Shikha"-----Here, this statement has 4 words and these words are called tokens.

✍In Chat GPT 3.5----limited to 1500 tokens.
if you upgrade Chat GPT 3.5 to 4.0----token limit increased to 2500.

✍2 types of Indexing--Forward Indexing and Backward indexing
Forward indexing starts with 0 and the direction is Left to right, whereas, Backward indexing starts with -1 and the
direction is right to left.

########Unable to add Image#######

✍we use print() to print multiple values at a time.

✍if you took multiple variables and assign some values to it and if you print without print function then you get the
last variable's value whatever you assign.

✍Also, if you use type() and pass the arguement with print() by passing variable to it then it will return the value
assigned to the variable not the type of the variable along with NoneType, Example will be available in below notes.

👇Let's understand Data types with Type Casting.
```

In [2]:
```python
import sys
import keyword
import operator
```

In [3]:
```python
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif',
'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

For multiline Comments--- we should learn this concept because when we deals with NLP (Natural Language Processing) we deals with text and multiline statement, so there we use such concepts.

In [4]:
```python
'Working with
   best software industry'   ## will give an error
```

```
  Cell In[4], line 1
    'Working with
    ^
SyntaxError: unterminated string literal (detected at line 1)
```

In [5]:
```python
'''Working with
      best software industry''' ## will not show error as three inverted commas is for printing multiline comments
```

Out[5]:
```
'Working with \n      best software industry'
```

In [16]:
```python
string_4= ('Lily '
          'pansy '
          'marrigold '
          'rose '
          'dahilia')
print(string_4)
```

```
Lily pansy marrigold rose dahilia
```

In [17]:
```python
string_6= 'shikha '
string_6= string_6*10
string_6
```

Out[17]:
```
'shikha shikha shikha shikha shikha shikha shikha shikha shikha shikha '
```

In [18]:
```python
len(string_6)
```

Out[18]:
```
70
```

Let's learn Forward and Backward indexing concepts

```
In [25]:  string_10 = 'Lets have fun with Python'
           string_10

Out[25]:  'Lets have fun with Python'

In [27]:  string_10[0]

Out[27]:  'L'

In [31]:  string_10[10] ##  because it runs on the concept of n-1 ---Forward Indexing

Out[31]:  'f'

In [33]:  string_10[5:9] ## will print <space> till letter 'v'----Forward Indexing--Also, :colon works in n-1 concept.

Out[33]:  'have'

In [34]:  string_10[-4] # will print from Right to Left

Out[34]:  't'

In [36]:  string_10

Out[36]:  'Lets have fun with Python'

In [42]:  string_10[-5:-1] #print reverse from 'n' to 't',,,, Here, starting index should be less than to the ending index if you
                             #reverse---Example-- -5 is less than -1

Out[42]:  'ytho'
```

Let's understand some facts about Complex Data Types.

```
In [43]:  comp = 9+2j
           comp

Out[43]:  (9+2j)

In [44]:  type(comp)

Out[44]:  complex

In [45]:  print(type(comp))

           <class 'complex'>

In [46]:  id(comp)

Out[46]:  1951372703312

In [47]:  len(comp) ### complex data type has no length.

           ---------------------------------------------------------------------------
           TypeError                                 Traceback (most recent call last)
           Cell In[47], line 1
           ----> 1 len(comp)

           TypeError: object of type 'complex' has no len()


In [48]:  comp.conjugate ###

Out[48]:  <function complex.conjugate()>

In [49]:  comp.imag

Out[49]:  2.0
```

```
In [50]: comp.real

Out[50]: 9.0
```

Let's deal with Integer value

```
In [52]: int(6666.90)

Out[52]: 6666
```

```
In [53]: int(99.55.33) ## cannot pass value which has 3 dots.

  Cell In[53], line 1
    int(99.55.33)
        ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

Let's introduce boolean to int with And, Or operators

```
In [55]: int(True) and int(True)

Out[55]: 1

In [56]: int(False) and int(False)

Out[56]: 0

In [57]: int(False) and int(True)

Out[57]: 0

In [58]: int(True) and int(False)

Out[58]: 0

In [59]: int(True) or int(True)

Out[59]: 1

In [60]: int(False) or int(False)

Out[60]: 0

In [61]: int(False) or int(True)

Out[61]: 1

In [62]: int(True) or int(False)

Out[62]: 1

In [89]: int("30")  ## will work coz, at last we pass numbers as an argument.

Out[89]: 30

In [90]: int("Thirty")  ## will not work coz, at last we are passing string as an argument.

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[90], line 1
----> 1 int("Thirty")

ValueError: invalid literal for int() with base 10: 'Thirty'
```

understanding Int Type casting with Boolean Datatype combining Exception Handling along with '<' operator

In [70]:

```
x = "shikha"
y = int(False)
try:
    if x<y:
        print("Oops not a valid one")
except:
    print("Please enter valid inputs")
finally:
    print("Code Executed")
```

```
Please enter valid inputs
Code Executed
```

In [73]:

```
x = 78
y = 56
try:
    if x<y:
        print("Oops not a valid one")
    else:
        print("Valid")
except:
    print("Please enter valid inputs")
finally:
    print("Code Executed")
```

```
Valid
Code Executed
```

In [77]:

```
x = int(True)
y = int(False)
try:
    if x<y:
        print("Oops not a valid one")
    else:
        print("Valid")
except:
    print("Please enter valid inputs")
finally:
    print("Code Executed")
```

```
Valid
Code Executed
```

In [75]:

```
x = int(False)
y = int(True)
try:
    if x<y:
        print("Oops not a valid one")
    else:
        print("Valid")
except:
    print("Please enter valid inputs")
finally:
    print("Code Executed")
```

```
Oops not a valid one
Code Executed
```

In [78]:

```
x = int(True)
y = int(False)
try:
    if x<y:
        print("Oops not a valid one", x)
    else:
        print("Valid", y)
except:
    print("Please enter valid inputs")
finally:
    print("Code Executed")

## In this the value of "False" will print because we use 'x' along with print statement.
```

```
Valid 0
Code Executed
```

In [79]:
```
x = int(True)
y = float(False)
try:
    if x<y:
        print("Oops not a valid one", x)
    else:
        print("Valid", y)
except:
    print("Please enter valid inputs")
finally:
    print("Code Executed")

### In this code the value 0.0 is printing because we assigned Float value to the 'y'
```
```
Valid 0.0
Code Executed
```

Let's revise little bit Data types again

In [7]:
```
val1 = 13
val2 = 12
print(val1)
print(val2)
```
```
13
12
```

In [10]:
```
type(print(val1))
```
```
13
```
Out[10]:
```
NoneType
```

In [9]:
```
type(val1)
```
Out[9]:
```
int
```

In [11]:
```
print(type(val1))
```
```
<class 'int'>
```

##### if you use type() and pass the arguement with print() by passing variable to it then it will always return the value assigned to the variable not the type of the variable along with NoneType

In [12]:
```
type(print(val1))
```
```
13
```
Out[12]:
```
NoneType
```

In [13]:
```
id(val1)  ##shows the address of the Val1
```
Out[13]:
```
140703279977640
```

In [15]:
```
## suppose if you take some variables and assign same values such as --###
val1 = 13
val2 = 13
print(id(val1))
print(id(val2))

## this will give you the same address, coz Python has a feature called Memory Management as it saves the space.
## In this example 2 variables pointing to the same address and that's where it is saving the space.
## In system or in python "Constructor" assigns the memory space to the values of the variables.
```
```
140703279977640
140703279977640
```

In [80]:
```
int("Excellent Python") ## Error because we cannot pass string as a value if we pass string as an arguement.
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[80], line 1
----> 1 int("Excellent Python")

ValueError: invalid literal for int() with base 10: 'Excellent Python'
```

In [81]:
```
int(5+19j) ##Error because we cannot pass Complex as a value if we pass complex as an arguement.
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[81], line 1
----> 1 int(5+19j)

TypeError: int() argument must be a string, a bytes-like object or a real number, not 'complex'
```

#### Let's deal with Float Datatypes.

In [82]:
```
float(50)
```
Out[82]:
```
50.0
```

In [83]:
```
float(True)
```
Out[83]:
```
1.0
```

In [84]:
```
float(1+9j)  ##Float will not accept complex Values as an arguement.
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[84], line 1
----> 1 float(1+9j)

TypeError: float() argument must be a string or a real number, not 'complex'
```

In [86]:
```
float('400') ## will work coz, at last we pass numbers as an argument.
```
Out[86]:
```
400.0
```

In [87]:
```
float("Four Hundred") ## will not work coz, at last we are passing string as an argument.
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[87], line 1
----> 1 float("Four Hundred")

ValueError: could not convert string to float: 'Four Hundred'
```

Let's introduce boolean to Float with And, Or operators

In [91]:
```
float(True) and float(True)
```
Out[91]:
```
1.0
```

In [92]:
```
float(True) and float(False)
```
Out[92]:
```
0.0
```

In [93]:
```
float(False) and float(True)
```
Out[93]:
```
0.0
```

In [94]:
```
float(False) and float(False)
```
Out[94]:
```
0.0
```

In [95]:
```
float(True) or float(True)
```
Out[95]:
```
1.0
```
In [96]:
```
float(True) or float(False)
```
Out[96]:
```
1.0
```
In [97]:
```
float(False) or float(True)
```
Out[97]:
```
1.0
```
In [98]:
```
float(False) or float(False)
```
Out[98]:
```
0.0
```

#### understanding Float Type casting with Boolean Datatype combining Exception Handling along with "!=" and logical operators

In [99]:
```
N = float(False)
Y = float(True)
try:
    if N != Y:
        print("Not equal to", N , Y)
    else:
        print("Equal to", N , Y)
except:
    print("Enter Valid inputs")
finally:
    print("Code is running perfectly")
```
```
Not equal to 0.0 1.0
Code is running perfectly
```
In [100]:
```
N = float(True)
Y = float(True)
try:
    if N != Y:
        print("Not equal to", N , Y)
    else:
        print("Equal to", N , Y)
except:
    print("Enter Valid inputs")
finally:
    print("Code is running perfectly")
```
```
Equal to 1.0 1.0
Code is running perfectly
```
In [103]:
```
N = float(True)
Y = float(True)
B = float(False)
U = float(True)
try:
    if N != Y and B <= U:
        print("Not equal to", N , Y , B , U)
    else:
        print("Equal to", N , Y , B , U)
except:
    print("Enter Valid inputs")
finally:
    print("Code is running perfectly")
```
```
Equal to 1.0 1.0 0.0 1.0
Code is running perfectly
```

In [105]:
```python
N = float(True)
Y = float(False)
B = float(True)
U = float(False)
try:
    if N != Y and B <= U:
        print("Not equal to", N , Y , B , U)
    else:
        print("Equal to", N , Y , B , U)
except:
    print("Enter Valid inputs")
finally:
    print("Code is running perfectly")
```

```
Equal to 1.0 0.0 1.0 0.0
Code is running perfectly
```

In [106]:
```python
N = float(True)
Y = float(False)
B = float(True)
U = float(False)
try:
    if N != Y or B <= U:
        print("Not equal to", N , Y , B , U)
    else:
        print("Equal to", N , Y , B , U)
except:
    print("Enter Valid inputs")
finally:
    print("Code is running perfectly")
```

```
Not equal to 1.0 0.0 1.0 0.0
Code is running perfectly
```

#### understanding Complex Type casting with Boolean, Float, Integer and string Datatype combining Exception Handling along with '>', '==', 'and' and 'or' operator

In [108]:
```python
com = 8+4j
com1 = 3+2j
print(com)
print(com1)
```

```
(8+4j)
(3+2j)
```

In [109]:
```python
com = 8+4j
com1 = 3+2j
com
com1
```

Out[109]:
```
(3+2j)
```

In [110]:
```python
com = 8+4j ## by default it is returning float values.
com.real
```

Out[110]:
```
8.0
```

In [115]:
```python
int(com.real)
```

Out[115]:
```
8
```

In [116]:
```python
int(com.imag)
```

Out[116]:
```
4
```

In [122]:
```
com = 8+4j
com1 = 3+2j

try:
    if com.real > com1.real:
        print("Greater number", int(com.real))
    else:
        print("Smaller number", int(com1.real))
except:
    ("print done")
finally:
    ("Code has been executed")
```

```
Greater number 8
```

In [125]:
```
com = 8+4j
com1 = 3+2j

try:
    if com.real < com1.imag:
        print("Greater number", int(com.real))
    else:
        print("Smaller number", int(com1.real))
except:
    ("print done")
finally:
    ("Code has been executed")
```

```
Smaller number 3
```

In [126]:
```
com = 8+4j
com1 = 3+2j

try:
    if com.imag == com1.imag:
        print("Greater number", int(com.imag))
    else:
        print("Smaller number", int(com1.imag))
except:
    ("print done")
finally:
    ("Code has been executed")
```

```
Smaller number 2
```

In [4]:
```
com = "Mahadev"
com1 = 3+2j

try:
    if com & com1:
        print("Greater number", com)
    else:
        print("Smaller number", com1)
except:
    print("done")
finally:
    print("Code has been executed")

## Why except will print because try block has an error , as "&" symbol used.
```

```
done
Code has been executed
```

let's jump to basic of complex again to understand better.

In [5]:
```
complex(10,4)
```

Out[5]:
```
(10+4j)
```

```
In [7]:  complex('eleven', 4) ## Complex cannot work with Strings.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[7], line 1
----> 1 complex('eleven', 4)

TypeError: complex() can't take second arg if first is a string
```

let's understand string better

```
In [8]:  str(3.9)
```

```
Out[8]: '3.9'
```

```
In [9]:  str('twelve')
```

```
Out[9]: 'twelve'
```

```
In [10]:  str(twelve) ## cannot pass string as an arguement without commas
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[10], line 1
----> 1 str(twelve)

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

```
In [11]:  str(2+7j)
```

```
Out[11]: '(2+7j)'
```

understanding string with basic swapping Program by combining Exception Handling

```
In [17]:  string_1 = 'amrita'
          string_2 = 'Singh'
          string_3 = string_2 , string_1
          string_3

          try:
              if string_3 == string_2:
                  print("matched")
              else:
                  print("not matched")
          except:
              print("Error in code")
          finally:
              print("Swapped", string_3)
```

```
not matched
Swapped ('Singh', 'amrita')
```

understanding Boolean with logical operators by combining Exception Handling

```
In [18]:  bool(True)
```

```
Out[18]: True
```

```
In [19]:  bool(False)
```

```
Out[19]: False
```

In [21]:

```
w = 56
q = 87

w > q & q == w
```

Out[21]:

```
False
```

In [22]:

```
w = 56
q = 87

try:
    if q != w & w <= q:
        print("Not equal to")
    else:
        print("Equal to")
except:
    print("Enter Valid inputs")
finally:
    print("Code is running perfectly")
```

```
Not equal to
Code is running perfectly
```

In [ ]: