## Data types

- integer

  int
- float

  float
- strings

  str
- boolean

  bool
- complex conjugate

  complex

*integer*

In [1]:
```
number=100
number
```

Out[1]:
```
100
```

In [2]:
```
# type of the varible :  type
type(number)
```

Out[2]:
```
int
```

# In maths we have different number system

- decimal
- binary
- octal
- hexa

### Binary

- Bi mean 2
- so the base is also 2
- 0 1 2 3 4 5 6 7 8 9
- here will use only two digits: 0 and 1
- representation is : 0b<combinations of 0 and 1>
- 0b110 , 0B011101

In [3]:
```
0b111
# python output will come as decimal format only
```

Out[3]: 7

In [4]:
```
0b1111
```

Out[4]: 15

In [ ]:
```
# 3 digits = 4   2   1
# 4 digits = 8   4   2   1
# 5 digits=  16  8   4   2   1
**********************
4    2     1    decimal output
**********************
0    0     0      0
0    0     1      1
0    1     0      2
0    1     1      3
1    0     0      4
1    0     1      5
1    1     0      6
1    1     1      7

4= 4 times 0 and 4 times 1 will come
2= 2 times 0 and 2 time  1
1= 1 time  0 and 1  time  1

ON=1
Off=0
```

In [ ]:
```
    **********************
 8  4    2     1    decimal output
    **********************
 0   0    0     0      0
 0   0    0     1      1
 0   0    1     0      2
 0   0    1     1      3
 0   1    0     0      4
 0   1    0     1      5
 0   1    1     0      6
 0   1    1     1      7
 1   0    0     0      8
 1   0    0     1      9
 1   0    1     0      10
 1   0    1     1      11
 1   1    0     0      12
 1   1    0     1      13
 1   1    1     0      14
 1   1    1     1      15
```

In [5]:
```
0b1110
```

Out[5]: 14

In [6]:
```
0b1001
```

Out[6]: 9

**Octa**

- octa mean 8
- so the base is also 8
- 0 1 2 3 4 5 6 7 8 9
- here will use only 8 digits: 0,1,2,3,4,5,6,7 (0 to 7 means 8 digits)
- representation is : 0o<combinations of 0 to 7>
- 0o123 , 0O7564

In [7]:
```
0o123
```

Out[7]: 83

In [8]:
```
0O756
```

Out[8]: 494

In [9]:
```
0o565
```

Out[9]: 373

**hexa**

- hexa mean 16
- so the base is also 16
- 0 1 2 3 4 5 6 7 8 9 , A ,B, C, D, E, F
- here will use only 16 digits: (0 to 9 and A to F)
- representation is : 0x<0 to 9 and A to F>
- 0xabc , 0X9DF

In [11]:
```
0xabc
```

Out[11]: 2748

*float*

In [13]:
```
number1=123.5
number1
```

Out[13]: 123.5

In [14]:
```
type(number1)
```

Out[14]: float

*e represntation*

In [15]:
```
10e3   #  10 * 1000 = 10000
```

Out[15]:
```
10000.0
```

In [16]:
```
10e+3  # 10e3  and 10e+3  both are same
```

Out[16]:
```
10000.0
```

In [17]:
```
1e5   # 1* 100000
```

Out[17]:
```
100000.0
```

In [18]:
```
10e-3   # 10/1000= 1/100=0.01
```

Out[18]:
```
0.01
```

In [ ]:
```
# ML ===== output as zero
# 0.00000000000000000000123
# 123e-10= 123/100000000000= 0.0000000000123
```

In [20]:
```
1000e-5  # 1000/100000
```

Out[20]:
```
0.01
```

In [23]:
```
10e-6
```

Out[23]:
```
1e-05
```

*Strings*

In [24]:
```
name="python"  # write any thing in quotes
type(name)
```

Out[24]:
```
str
```

In [25]:
```
name1="100"
type(name1)
```

Out[25]:
```
str
```

In [26]:
```
name2='how are you'
name2
```

Out[26]:
```
'how are you'
```

In [ ]:
```
hai

how are you

write above sentence in the one string
when you call your output should line be lie
/n   /t
```

In [27]:
```
name3="hai
       how are you"

name3
```

```
  Cell In[27], line 1
    name3="hai
          ^
SyntaxError: unterminated string literal (detected at line 1)
```

In [28]:
```
name4='hai
       how are you'

name4
```

```
  Cell In[28], line 1
    name4='hai
          ^
SyntaxError: unterminated string literal (detected at line 1)
```

In [29]:
```
# whenever you want write multi lines  use triple quotes

name5="""hai
         how are you"""
name5

# doc string
```

Out[29]:
```
'hai\n         how are you'
```

In [ ]:
```
# doc string is very important
# jupyter notebook : markdown option
# vs code pycharm :  doc string
```

In [30]:
```
import random
random.randint()
```

my name is python

In [ ]:
```
"""my name is python"""
```

In [31]:
```
"""
i have taken 20 in a
i have taken 30 in b
im cal
"""
a=20 #
b=30
a+b
```

Out[31]:
```
50
```

In [ ]:
```
import random
random.randint()
```

In [ ]:
```
# Integer
    # Binary
    # Ocatl
    # hexa

# Float
# String
   # single quotes
   # double quotes
   # triple quotes
```

In [1]:
```
string1='python'
string1
```

Out[1]: 'python'

In [2]:
```
string2="good morning"
string2
```

Out[2]: 'good morning'

In [ ]:
```
# triple quotes ====== doc string
# you are writing some information
string3="""how are you"""
```

**Boolean**

In [ ]:
```
# whenever if you ask any question to the computer
# what its repaly? True or False
# True=1
# False=0

True  # keyword
False # keyword
```

In [3]:
```
value= False # we are assigning False in a variable value
value
```

Out[3]: False

In [4]:
```
type(value)
```

Out[4]: bool

In [5]:
```
valu1=True
type(valu1)
```

Out[5]: bool

In [ ]:
```
name1=True  # boolean
name2=true  # both are variables
name3="True" # string
```

In [6]:
```
name2=true

# Have you defined True before?
# i did not defiend True, thats whay Im not able to use that
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call las
t)
Cell In[6], line 1
----> 1 name2=true

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

In [7]:
```
true=100    # 100 is saving in a variabel true
name2=true  # true is saving in a variable name2
name2       # so name2 =100
```

Out[7]:
```
100
```

In [8]:
```
a=100   # 100 saved in a
b=200   # 200 saved in b
a=b     # b is saved in a=200  100=200
b=a     # a is saved in b=200
a=500   # 500 saved in a=500
b=a     # a is saved in b=500
b=900   # 900 is saved in b=900
a       # latest value =500


100 ---- > 200 ---- > 500
200 ----- > 200 ---- > 500 ---- >900
```

Out[8]:
```
500
```

In [9]:
```
b
```

Out[9]:
```
900
```

In [ ]:
```
left=right
100=200
```

In [10]:
```
100=200
```

```
  Cell In[10], line 1
    100=200
    ^
SyntaxError: cannot assign to literal here. Maybe you meant '==' instead o
f '='?
```

**Complex-Conjugate**

- a+jb or a+ib
- where a= real number
- b= imaginary number
- i= sqrt(-1)

In [11]:
```
number= 3+5j
number
```

Out[11]: (3+5j)

In [12]:
```
type(number)
```

Out[12]: complex

In [13]:
```
number1= 3-5j
number1
```

Out[13]: (3-5j)

In [ ]:
```
# If i want to retrive only real value seperately
3+5j
# 3 is the real value
# 5 is the imaginry value
```

In [14]:
```
number=3+5j
number
```

Out[14]: (3+5j)

In [15]:
```
dir(number)
```

Out[15]:
```
['__abs__',
 '__add__',
 '__bool__',
 '__class__',
 '__complex__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmul__',
 '__rpow__',
 '__rsub__',
 '__rtruediv__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 'conjugate',
 'imag',
 'real']
```

In [16]:
```
number.real   # <pname>.<method>
```

Out[16]:
```
3.0
```

In [17]:
```
number.imag
```

Out[17]:
```
5.0
```

In [19]:
```
number.conjugate()

# +  become  -
# -  become  +
```

Out[19]:
```
(3-5j)
```

In [22]:
```
number.conjugate()
```

Out[22]:
```
<function complex.conjugate()>
```

In [ ]:
```
number1=3-5j
# number1.real = 3
# number1.imag = -5
# number1.conjgate= 3+5j
```

In [ ]:
```
3-5j  ====== > 3   ,
```

In [20]:
```
complex(4,10)
# real=4
# image=10
# 4+10j
```

Out[20]:
```
(4+10j)
```

In [21]:
```
complex(4,-10)
# Inside bracket what you are seeing
# parameters/arguments
```

Out[21]:
```
(4-10j)
```

In [ ]:
```
complex()  # if i did not provide anything inside
```

- whenever if you see any barckets (), that is called function
- another name of function is called method
- inside brackets what you are seeing is called parameters
- another name for parameters is arguments

In [32]:
```
import random
random.randint(1,100)
```

Out[32]:
```
88
```

In [28]:
```
import math
math.sin()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
t)
Cell In[28], line 2
      1 import math
----> 2 math.sin()

TypeError: math.sin() takes exactly one argument (0 given)
```

In [33]:
```
complex()

# deafault arguments

# if you not provide anything inside bracket
# I will use real value as 0
# imaginary value also 0
# 0+0j  =====> 0j only
```

Out[33]:
```
0j
```

In [ ]:
```
A) 0+0j
B) 0j
C) 0-0j
D) 0
E) error
```

In [ ]:
```
A) random.randint() # error  is it necessary to provide arguments
B) math.sin()       # error  is it necesaary to provide arguments
C) complex()        # no error
```

In [ ]:
```
random.randint(a, b)
math.sin(x, /)
complex(real=0, imag=0)
```

In [34]:
```
random.randint()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last
t)
Cell In[34], line 1
----> 1 random.randint()

TypeError: Random.randint() missing 2 required positional arguments: 'a' a
nd 'b'
```

In [35]:
```
math.sin()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[35], line 1
----> 1 math.sin()

TypeError: math.sin() takes exactly one argument (0 given)
```

In [36]:
```
complex()
```

Out[36]:
```
0j
```

In [ ]:
```
# packages
# methods
# parameters
```

In [ ]:
```
- int

- float

- str

- bool

- complex
```