Data Types

- integer int
- float float
- string str
- boolean bool

integer

- binary
- octa
- hexa

binary

- bi means two
- it require only two digits
- we have 0 1 2 3 4 5 6 7 8 9
- to make binary we need ony 0 and 1
- example: 0b111, 0B111

In [2]:
```
0b111
```

Out[2]: 7

In [3]:
```
0B111
```

Out[3]: 7

In [4]:
```
0b101
```

Out[4]: 5

In [ ]:
```
4 2 1
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
```

In [5]:
```
0b1111
```

Out[5]: 15

octa

- OCTA means eight
- it require only 8 digits
- we have 0 1 2 3 4 5 6 7 8 9
- to make binary we need ony 0 1 2 3 4 5 6 7 8 9
- example: 0o776, 0o1234

In [10]:
```
0o123
```

Out[10]: 83

Hexa

- Hexa means 16
- it require only 0 to 9 and A to F.
- we have 0 1 2 3 4 5 6 7 8 9
- to make binary we need ony 0 1 2 3 4 5 6 7 8 9 A B C D E F
- example: 0X7A62, 0xABC

In [13]:
```
0X6A1
# 16^0*1+16^1*(10)+16^2*6
```

Out[13]: 1697

float

In [14]:
```
number = 10.56
type(number)
```

Out[14]: float

In [15]:
```
10e1
# 10 is multiplying with e1= 10
```

Out[15]: 100.0

In [16]:
```
10e2
# 10 is multiplying with e2= 100
```

Out[16]: 1000.0

In [17]:
```
10e3
# 10 is multiplying with e3= 1000
```

Out[17]: 10000.0

- e1 e2 e3 e4 means 10 is multiplied with those many zer0s

In [20]:
```
10e-1
#10 is divided by 10 = 10/10
10e-1  #10/10
10e-2  #10/100
10e-3  #10/1000
10e-4  #10/10000
10e+5  #10*100000
12345e-2 #12345/100
```

Out[20]: 123.45

In [21]:
```
12345e-10
#check the output is zero
```

Out[21]: 1.2345e-06

- 10e2 as same as 10e+2 == Multiplying
- 10e-2 means dividing

String

In [ ]:
```
integer #variables
int      #keyword
"integer" #string
```

In [23]:
```
name="python"
name
```

Out[23]: 'python'

In [24]:
```
type(name)
```

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

In [26]:
```
print('hello "python"')
```

```
hello "python"
```

In [30]:
```
print("hello 'python'")
```

```
hello 'python'
```

TripleQuotes

- Triple quotes are used to write a story of a specific code of string called doc string
- Doc string
- if somebody is writing the string in triple quotes then they are trying to explain teh code to the engineer

In [31]:
```
import random
```

In [32]:
```
random.randint(2,3)
```

Out[32]:
```
3
```

In [ ]:
```
# int
# float
# str
```

boolean

In [34]:
```
value1= False
type(value1)
```

Out[34]:
```
bool
```

In [35]:
```
value2 ="True"
type(value2)
```

Out[35]:
```
str
```

- Not defined error is very common as beginner

Complex

- It represent a a+bj
- where a is real number

- b is an imaginary number
- ex: 3+4j

In [39]:
```
value = 3+4j
type(value)
```

Out[39]:
```
complex
```

In [40]:
```
dir(value)
```

Out[40]:
```
['__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 [ ]:
```
# valu is kind of a package
# m-1:conjugate
# m-2:imag
# m-3:real
```

In [ ]:
```
# pacakgename.method name
```

In [42]:
```
value = 3+4j
dir(value)
#conjugate imag real
value.conjugate()
```

Out[42]:
```
(3-4j)
```

In [43]:
```
value.real
```

Out[43]:
```
3.0
```

In [44]:
```
value.imag
```

Out[44]:
```
4.0
```

In [45]:
```
value.omkar
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[45], line 1
----> 1 value.omkar

AttributeError: 'complex' object has no attribute 'omkar'
```

- no kid error(parent doesn't have that kid) is the attribute error very common error of the student

Errors

- Syntax error must be avoided
- name error
- attribute error

In [ ]:
```

```