(3.1) Numbers

1. Integers
2. Float Point Numbers
3. Complex Numbers
4. Binary Numbers
5. Octal Numbers
6. Decimal Numbers
7. Hexa Decimal Numbers
8. Positive Numbers
9. Negative Numbers

Everything is Object , Dynamic Language

1. int data type:

We can use int data type to represent whole numbers (integral values)
Eg:

In [22]:
```
a=10
print(type(a)) #int
a=-10
print(type(a)) #int
```

```
<class 'int'>
<class 'int'>
```

Note:

In Python2 we have long data type to represent very large integral values.
But in Python3 there is no long type explicitly and we can represent long values also by using int type only.

We can represent int values in the following ways

1. Decimal form
2. Binary form
3. Octal form
4. Hexa decimal form

In [4]:
```
x = 999999999999999999999999999999999999999999999999
```

In [5]:
```
print(x)
print(type(x))
```

```
999999999999999999999999999999999999999999999999
<class 'int'>
```

1. Decimal form(base-10):

It is the default number system in Python
The allowed digits are: 0 to 9
Eg: a =10

2. Binary form(Base-2):

```
The allowed digits are : 0 & 1
Literal value should be prefixed with 0b or 0B
Eg: a = 0B1111
    a = 0B123
    a = b111
```

3. Octal Form(Base-8):

```
The allowed digits are : 0 to 7
Literal value should be prefixed with 0o or 0O.
Eg: a=0o123
    a=0o786
```

4. Hexa Decimal Form(Base-16):

```
The allowed digits are : 0 to 9, a-f (both lower and upper cases are allowed)
Literal value should be prefixed with 0x or 0X
Eg:
 a =0XFACE
 a =0XBeef
 a =0XBeer
```

Note:

```
Being a programmer we can specify literal values in decimal, binary, octal and hexa
decimal forms. But PVM will always provide values only in decimal form.
```

In [3]:

```
a=10
b=0o10
c=0X10
d=0B10
print(a)#10
print(b)#8
print(c)#16
print(d)#2
```

```
10
8
16
2
```

In [20]:

```
print(int("1111", 10))
print(int("1111", 2))
print(int("1111", 8))
print(int("1111", 16))
```

```
1111
15
585
4369
```

Base Conversions

```
Python provide the following in-built functions for base conversions
```

1.bin():

We can use bin() to convert from any base to binary
Eg:

In [6]:
```
bin(15)
```

Out[6]:
```
'0b1111'
```

In [7]:
```
bin(0o11)
```

Out[7]:
```
'0b1001'
```

In [8]:
```
bin(0X10)
```

Out[8]:
```
'0b10000'
```

2. oct():

We can use oct() to convert from any base to octal
Eg:

In [9]:
```
oct(10)
```

Out[9]:
```
'0o12'
```

In [10]:
```
oct(0B1111)
```

Out[10]:
```
'0o17'
```

In [11]:
```
oct(0X123)
```

Out[11]:
```
'0o443'
```

3. hex():

We can use hex() to convert from any base to hexa decimal
Eg:

In [12]:
```
hex(100)
```

Out[12]:
```
'0x64'
```

In [13]:
```
hex(0B111111)
```

Out[13]:
```
'0x3f'
```

In [14]:
```
hex(0o12345)
```

Out[14]:
```
'0x14e5'
```

2. float data type:

We can use float data type to represent floating point values (decimal values)

In [23]:
```
f=1.234
type(f)
```

Out[23]:
```
float
```

In [13]:
```
print( type( 56.67 ) )
```

```
<class 'float'>
```

We can also represent floating point values by using exponential form (scientific notation)

The main advantage of exponential form is we can represent big values in less memory.

In [24]:
```
f=1.2e3 #instead of 'e' we can use 'E'
```

In [25]:
```
print(f, type(f))
```

```
1200.0 <class 'float'>
```

Note: We can represent int values in decimal, binary, octal and hexa decimal forms. But we can represent float values only by using decimal form.

In [26]:
```
f=0B11.01
```

```
  File "C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10260/1390421556.py", line 1
    f=0B11.01
          ^
SyntaxError: invalid syntax
```

In [27]:
```
f=0o123.456
```

```
  File "C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10260/3650593558.py", line 1
    f=0o123.456
          ^
SyntaxError: invalid syntax
```

In [28]:
```
f=0X123.456
```

```
  File "C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10260/2688966689.py", line 1
    f=0X123.456
          ^
SyntaxError: invalid syntax
```

3. complex data type:

a + bj

a and b contain integers or floating point values

In [33]:
```
a = (3+5j)
```

```
b = (3-5j)
c = 10+5.5j
d = 0.5+0.1j
```

In [34]:
```
print(type(a), a)
print(type(b), b)
print(type(c), c)
print(type(d), d)
```

```
<class 'complex'> (3+5j)
<class 'complex'> (3-5j)
<class 'complex'> (10+5.5j)
<class 'complex'> (0.5+0.1j)
```

In the real part if we use int value then we can specify that either by decimal,octal,binary
or hexa decimal form.
But imaginary part should be specified only by using decimal form.

In [35]:
```
a=0B11+5j
print(a)
```

```
(3+5j)
```

In [37]:
```
a=3+0B11j
print(a)
```

```
  File "C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10260/1730358453.py", line 1
    a=3+0B11j
            ^
SyntaxError: invalid syntax
```

Even we can perform operations on complex type values.

In [38]:
```
a = (3+5j)
b = (3-7j)
print(a+b)
```

```
(6-2j)
```

Note:

Complex data type has some inbuilt attributes to retrieve the real part and imaginary part

We can use complex type generally in scientific Applications and electrical engineering
Applications.

In [39]:
```
c=10.5+3.6j
print(c.real) #10.5
print(c.imag) #3.6
```

```
10.5
3.6
```

In [ ]:

In [ ]: