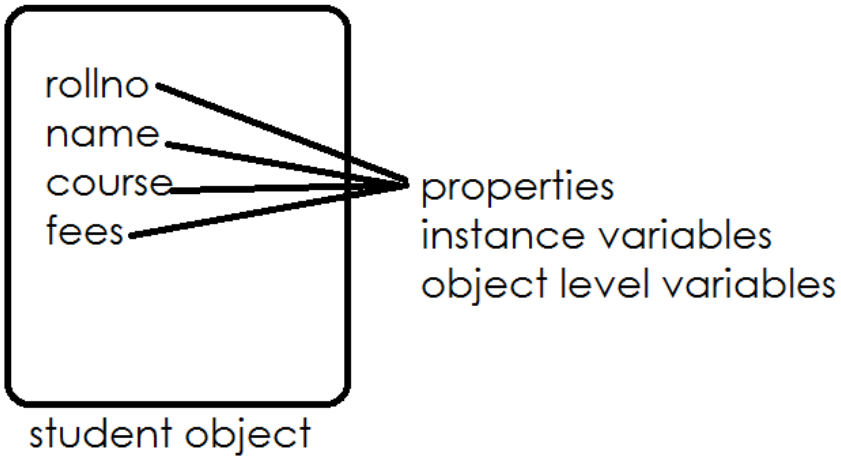
Object level variables or instance variables

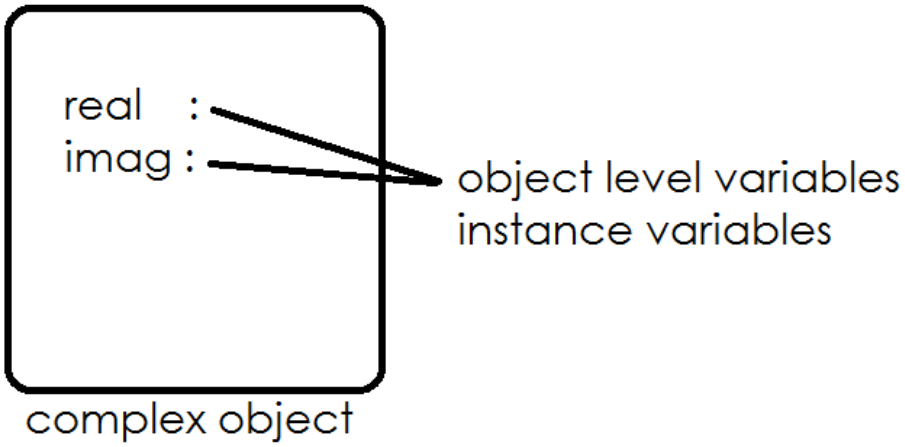
Every object is having properties

These properties are defined using object level variables or instance variables.

We can create instance variables or object level variables in different ways

1. Inside constructor
2. setattr/getattr predefined functions
3. properties class





Object level variables or instance variables memory is allocated on creation of object.

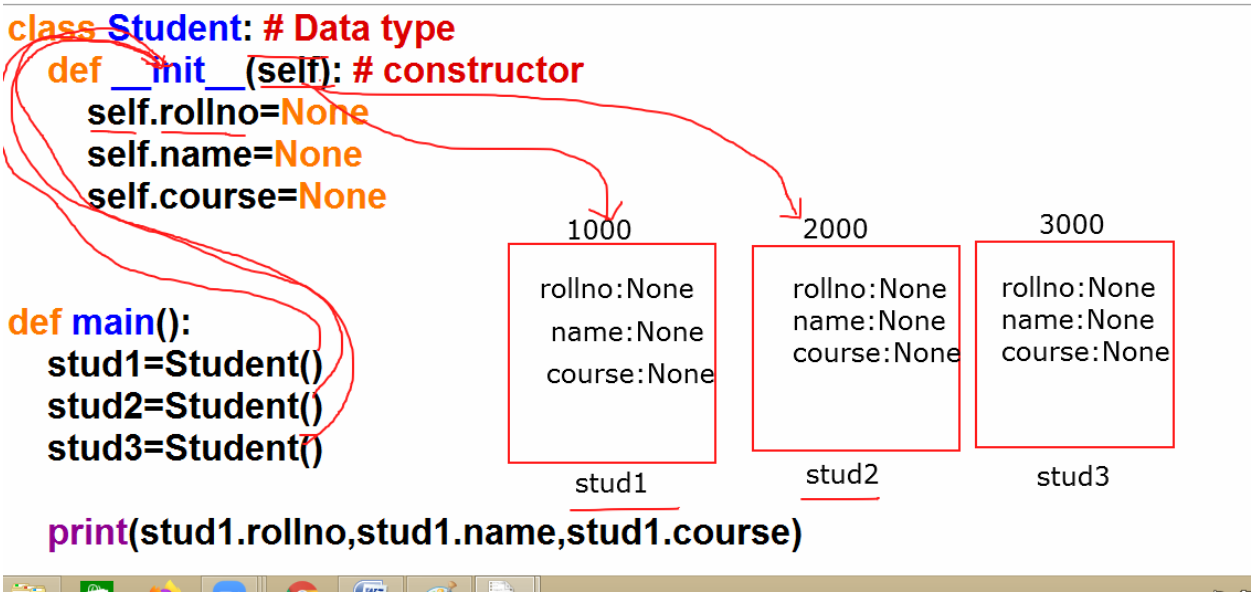
These variables bind with object name

We cannot use variables without creating object

Object level variables bind with “self”, inside class

Object level variables can be used only inside object level methods within class.

Creating object level variables within constructor method



Example:

```
class Student: # Data type
    def __init__(self): # constructor
        self.rollno=None
        self.name=None
        self.course=None


def main():
    stud1=Student()
    stud2=Student()
    stud3=Student()

    print(stud1.rollno,stud1.name,stud1.course)
    print(stud2.rollno,stud2.name,stud2.course)
    print(stud3.rollno,stud3.name,stud3.course)
    comp1=complex()
    print(comp1.real,comp1.imag)
    comp2=complex()
    print(comp2.real,comp2.imag)
main()
```

Output:

```
None None None
None None None
None None None
0.0 0.0
0.0 0.0
>>>
```

Example:

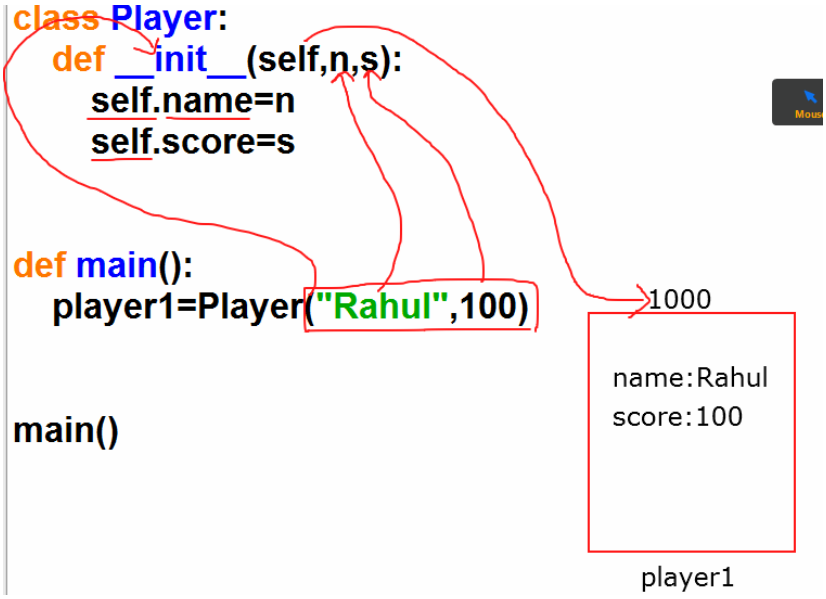
```
class Point:
    def __init__(self):
        self.x=100
        self.y=200
def main():
    comp1=complex()
    print(comp1.real)
    print(comp1.imag)
    point1=Point()
    point2=Point()
    print(point1.x,point1.y)
    print(point2.x,point2.y)
main()
```

Output:

```
0.0
0.0
100 200
100 200
```

Constructor with arguments

Constructor with arguments receive initial values of object



Example:

```
class Player:
    def __init__(self,n,s):
        self.name=n
        self.score=s


def main():
    player1=Player("Rahul",100)
    player2=Player("Virat",70)
    print(player1.name,player1.score)
    print(player2.name,player2.score)


main()
```

Output:

```
Rahul 100
Virat 70
>>>
```

 # write a program to read the scores of n players

```
class Player:
```

```
    def __init__(self,n,s):
        self.name=n
        self.score=s
def main():
    players_list=[]
    n=int(input("enter how many players?"))
    for i in range(n):
        name=input("Enter Name")
        score=int(input("Enter Score"))
        p=Player(name,score)
        players_list.append(p)
    total=0
    for player in players_list:
        print(f'{player.name}\t{player.score}')
        total=total+player.score

     print(f'total score {total}')
main()
```

Output:

```
enter how many players?2
Enter Namerahul
Enter Score120
Enter Namevirat
Enter Score20
rahul  120
virat   20
total score 140
>>>
```

Example:

```
class A:
    def __init__(self):
        print("without argument constructor")
    def __init__(self,x):
        print("with argument constructor")

def main():
    obj1=A(100)
```

main()
Output:
with argument constructor
>>>

There is no overloading in python
If multiple methods are with same name, the old methods are replaced with new method
This can achieved using default arguments, variable length arguments and keyword arguments.