OOP

Polymorphism

- Duck Typing
- Operator Overloading
- Method Overriding

Using methods in other classes

OOP

```
class b:
        def k(self):
                print("This is k function")

class a:
        def a(self,obj2):
                obj2.k()

obj2=b()
obj = a()
obj.a(obj2)
```

Duck Typing

OOP

```
class b:
        def k(self):
                print("This is k function")

class a:
        def a(self,obj2):
                obj2.k()

class d:
        def k(self):
                print("This is k in d")


obj2=d()
obj = a()
obj.a(obj2)
```

OOP

Operator Overloading

$$\underbrace{5 \overset{\overset{\text{Operators}}{\uparrow}}{+} 2}_{\text{Operands}}$$

OOP

Operator Overloading

(Everything in python is a class)

```
a=4
b=5
c=a+b
print(c)

print(int.__add__(a,b))
```

OOP

Operator Overloading

(Int class has various methods)

+	• `__add__()`
-	• `__sub__()`
*	• `__mul__()`

Overloading Addition Operator

OOP

```
class a:
        def __init__(self,m1,m2):
                self.m1=m1
                self.m2=m2
        def __add__(obj1,obj2):
                x = obj1.m1+obj2.m1
                y = obj1.m2+obj2.m2
                z = a(x,y)
                return z


s1 = a(3,4)
s2 = a(44,55)

s3 = s1+s2
print(s3.m1)
```

OOP

Overloading Greater than Operator

```
class a:
        def __init__(self,m1,m2):
                self.m1=m1
                self.m2=m2
        def __gt__(obj1,obj2):
                x = obj1.m1+obj1.m2
                y = obj2.m1+obj2.m2
                if x>y:
                        return True
                else:
                        return False

s1 = a(3,4)
s2 = a(44,55)

if s1>s2:
        print("s1 wins")
else:
        print("s2 wins")
```

OOP

Method Overriding

```
class a:
        def greet(self):
                print("Welcome to class a")

class b(a):
        def greet(self):
                print("Welcome to class b")

obj = b()
obj.greet()
```

Iterator

OOP

```
a = [2,33,45,67,890,3]

c = iter(a)

print(c.__next__())

for i in a:
		print(c.__next__())
```

Generators Example 1

OOP

```
def hello():
        yield 1
        yield 2
        yield 3

values=hello()
print(values.__next__())
print(values.__next__())
print(values.__next__())
```

OOP

Generators Example 2

```
def sq():
        n=1
        while n<=10:
                yield n*n
                n+=1

values=sq()

print(next(values))
for i in values:
        print(i)
```