In [ ]:
```
#straightforward demonstration of polymorphism in Python
print(4+5)
print("4"+"5")
print("ab"+"cd")
```

```
9
45
abcd
```

In [ ]:
```
#operator overloading
a=3
b=5
print(a+b)
print(int.__add__(a,b))
```

```
8
8
```

In [ ]:
```
class stud:
  def name(self):
    print("hello")
o=stud()
print (o)
```

```
<__main__.stud object at 0x7f344f68f0d0>
```

In [ ]:
```
class stud:
  def name(self):
    print("hello")
  def __str__(self):
    return ( "your modifying ur print function")
o=stud()
print (o)
```

```
your modifying ur print function
```

In [ ]:
```python
#operator overloading
class batsman:
  def __init__(self,a,b):
    self.a=a
    self.b=b
  def __add__(self,other):
    sum1=self.a+other.a
    sum2=self.b+other.b
    print(sum1,sum2)
    bat3=batsman(sum1,sum2)
    return bat3
  def __str__(self):
    return "{} is the sum of first scores and {} is the sum of second
 scores".format(self.a,self.b)
bat1=batsman(40,50)
bat2=batsman(80,20)
bat3=bat1+bat2
print(bat3)
```

```
120 70
120 is the sum of first scores and 70 is the sum of second scores
```

In [ ]:
```python
#operator overloading ""&gt;symbol"
class student:
  def __init__(self,m1,m2,m3):
    self.m1=m1
    self.m2=m2
    self.m3=m3
  def __gt__(self,s1):
    sum1=self.m1+self.m2+self.m3
    sum2=s1.m1+s1.m2+s1.m3
    if (sum1>sum2):
      return True
    else:
      return False

stud1=student(30,40,50)
stud2=student(70,20,50)
if stud1>stud2:
  print ("stud1 wins")
else:
  print ("stud2 wins")
```

```
stud2 wins
```

In [ ]:
```
a=3
print(dir(a))
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__',
'__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float_
_', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribut
e__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__',
'__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or_
_', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__red
uce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '_
_rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift_
_', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr_
_', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv_
_', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator',
'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

In [ ]:
```
#operator overloading
class batsman:
  def __init__(self,a,b):
    self.a=a
    self.b=b
  def __add__(self,other):
    sum1=self.a+other.a
    sum2=self.b+other.b
    return sum1,sum2
bat1=batsman(40,50)
bat2=batsman(80,20)
3,bat4=bat1+bat2
print(bat3,bat4)
```

```
120 70
```

In [ ]:
```
#sub method
class man:
  def __init__(self,hgt):
    self.hgt=hgt
  def __sub__(self,s1):
    difference=self.hgt-s1.hgt
    return difference
man1=man(160)
man2=man(175)
dif=man1-man2
print(dif)
```

```
-15
```

In [ ]:
```python
#polymorphism in class methods
class India():
    def capital(self):
        print("New Delhi is the capital of India.")
    def language(self):
        print("Hindi is the most widely spoken language of India.")
class USA():
    def capital(self):
        print("Washington, D.C. is the capital of USA.")
     def language(self):
        print("English is the primary language of USA.")
obj_ind = India()
obj_usa = USA()
for country in (obj_ind, obj_usa):
    country.capital()
    country.language()
```

```
New Delhi is the capital of India.
Hindi is the most widely spoken language of India.
Washington, D.C. is the capital of USA.
English is the primary language of USA.
```

In [ ]:
```python
#method overriding
class Bird:
     def flight(self):
       print("Most of the birds can fly but some cannot")
class parrot(Bird):
     def flight(self):
       print("Parrots can fly")
class penguin(Bird):
     def flight(self):
      print("Penguins do not fly")
obj_bird = Bird()
obj_parr = parrot()
obj_peng = penguin()
obj_bird.flight()
obj_parr.flight()
obj_peng.flight()
```

```
Most of the birds can fly but some cannot
Parrots can fly
Penguins do not fly
```

In [ ]:
```
#method overloading
class bird:
  def brdclass(self,name=None):
    self.name=name
    if self.name=="parrot":
      print("can fly")
    if self.name=="penguin":
      print("cannt fly")
    if self.name==None:
      print("not a bird")
birdobj=bird()
birdobj.brdclass("parrot")
birdobj.brdclass()
```

```
can fly
not a bird
```

In [ ]:
```
#abstraction-hiding the information-giving access to the information n
eeded
from abc import ABC,abstractmethod
class computer(ABC):
  @abstractmethod
  def process(self):
    pass
c1=computer()
```

```
---------------------------------------------------------------------------
----
TypeError                                 Traceback (most recent call l
ast)
<ipython-input-14-c8c68480bb4c> in <module>()
      5   def process(self):
      6     pass
----> 7 c1=computer()

TypeError: Can't instantiate abstract class computer with abstract meth
ods process
```

In [ ]:
```
#abstraction
from abc import ABC,abstractmethod
class computer(ABC):
  @abstractmethod
  def process(self):
    pass
```

In [ ]:
```python
#abstraction
from abc import ABC,abstractmethod
class computer(ABC):
  @abstractmethod
  def process(self):
    pass
class laptop(computer):
  def process(self):
    print("its running")
comp1=laptop()
comp1.process()
```

```
its running
```

In [ ]:
```python
class X(object):
    def __init__(self, a):
        self.num = a
    def doubleup(self):
        self.num *= 2

class Y(X):
    def __init__(self, a):
        X.__init__(self, a)
    def tripleup(self):
        self.num *= 3

obj = Y(4)
print(obj.num)

obj.doubleup()
print(obj.num)

obj.tripleup()
print(obj.num)
```

```
4
8
24
```