Polymorphism

It refers to use of a single type entity(methods, operator) or object to represent the different types in different scenario.

In [1]:

```
class Bird:
    def intro(self):
        print("There are many types of birds")
    def fly(self):
        print("Most of the birds can fly but some cannot")

class sparrow(Bird):
    def fly(self):
        print("Sparrow can fly")

class ostrich(Bird):
    def fly(self):
        print("Ostrich cannot fly")

obj_bird = Bird()
obj_sparrow = sparrow()
obj_ostrich = ostrich()
```

In [2]:

```
obj_bird.intro()
obj_bird.fly()
```

```
There are many types of birds
Most of the birds can fly but some cannot
```

In [3]:

```
obj_sparrow.intro()
obj_sparrow.fly()
```

```
There are many types of birds
Sparrow can fly
```

In [4]:

```
obj_ostrich.intro()
obj_ostrich.fly()
```

```
There are many types of birds
Ostrich cannot fly
```

In [5]:

```
class Cat:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def info(self):
        print("I am a Cat. My name is {}. I am {} years old".format(self.name,self.age))
    def make_sound(self):
        print("Meow")

class Cow:
    def __init__(self,name,age):
        self.name = name
        self.age = age
    def info(self):
        print("I am a Cow. My name is {}. I am {} years old".format(self.name,self.age))
    def make_sound(self):
        print("Moo")

cat1 = Cat("Kitty",10)
cat2 = Cat("Pussy",15)
cow1 = Cow("Siddhi",15)
cow2 = Cow("Fluffy",12)
```

In [6]:

```
cat1.info()
cat1.make_sound()
cat2.info()
cat2.make_sound()
```

```
I am a Cat. My name is Kitty. I am 10 years old
Meow
I am a Cat. My name is Pussy. I am 15 years old
Meow
```

In [7]:

```
cow1.info()
cow1.make_sound()
cow2.info()
cow2.make_sound()
```

```
I am a Cow. My name is Siddhi. I am 15 years old
Moo
I am a Cow. My name is Fluffy. I am 12 years old
Moo
```

In [8]:

```
#Using for loop
for animal in (cat1, cow1):
    animal.info()
    animal.make_sound()
```

```
I am a Cat. My name is Kitty. I am 10 years old
Meow
I am a Cow. My name is Siddhi. I am 15 years old
Moo
```

In [9]:

```
from math import pi

class Shape:
    def __init__(self,name):
        self.name = name
    def Area(self):
        pass
    def Fact(self):
        return "I am two-dimensional shape"
    def __str__(self):
        return self.name

class Square(Shape):
    def __init__(self,length):
        super().__init__("Square")
        self.length = length
    def Area(self):
        return self.length**2
    def Fact(self):
        return "Squares have each angle equal to 90 degrees."


class Circle(Shape):
    def __init__(self,radius):
        super().__init__("Circle")
        self.radius = radius
    def Area(self):
        return pi*self.radius**2

a  = Square(4)
b = Circle(7)
print(b)
print(b.Fact())
print(a.Fact())
print(b.Area())
```

```
Circle
I am two-dimensional shape
Squares have each angle equal to 90 degrees.
153.93804002589985
```

Method Overloading

Methods in python can be called with zero, one or more parameters. This process of calling the same method in different ways is called method overloading. Overloading is a method that can do different functionalities with same name.

In [10]:

```
class VIP:
    def Overloading(self,x=None,y=None):
        if x==None and y == None:
            print("Nothing!")
        elif x!=None and y!=None:
            print("Addition of two numbers",x+y)
        else:
            print("You have passed only one argument :",x)
```

In [11]:

```
obj = VIP()
obj.Overloading()
```

```
Nothing!
```

Method Overriding

Method overriding allow a sub class or child class to provide a specific implementation of a method that is already provided by one of its super classes or parent classes.

In [12]:

```
class Animal:
    multicelular = True
    eukaryotic = True

    def breathe(self):
        print("I breathe oxygen")

    def feed(self):
        print("I eat food")

class Herbivorous(Animal):
    def feed(self):
        print("I eat only plants. I am vegetarian.")

class Omnivorous(Herbivorous):
    def feed(self):
        print("I eat both")

animal1 = Animal()
animal2 = Herbivorous()
animal3 = Omnivorous()
```

In [13]:

```
animal1.feed()
animal2.feed()
animal3.feed()
```

```
I eat food
I eat only plants. I am vegetarian.
I eat both
```

Difference between Overloading and Overriding

Overloading

1. Method with the same name but different number of arguments.
2. Inheritance is optional
3. Take place in method within the class
4. Can be done within a class.
5. Binding of overloaded method is done at compile time, hence it is a part of compile time polymorphism.
6. Static method can be overloaded.
7. Increase core reusability.
8. Relates with polymorphism.

Overriding

1. method with same name and same number of arguments.
2. inheritance is required/must.
3. Methods resides in different classes.
4. Atleast two classes are required.
5. Binding of overriden methods is done at runtime, hence it is part of runtime polymorphism.
6. Static methods cannot be overriden.
7. Used in implementation of specific scenarios.
8. It related with inheritance.

In [ ]:

```

```

In [ ]:

```

```