Polymorphism

Poly means many. Morphs means forms.
Polymorphism means 'Many Forms'.

1. Duck Typing Philosophy of Python:

In Python we cannot specify the type explicitly. Based on provided value at runtime the type will be considered automatically. Hence Python is considered as Dynamically Typed Programming Language.

```
def f1(obj):
    obj.talk()
```

What is the type of obj? We cannot decide at the beginning. At runtime we can pass any type.Then how we can decide the type?
At runtime if 'it walks like a duck and talks like a duck,it must be duck'. Python follows this principle. This is called Duck Typing Philosophy of Python.

Overloading:

We can use same operator or methods for different purposes.

There are 3 types of overloading
1. Operator Overloading
2. Method Overloading
3. Constructor Overloading

1. Operator Overloading:

We can use the same operator for multiple purposes, which is nothing but operator overloading.

Python supports operator overloading.

We can overload + operator to work with Book objects also. i.e Python supports Operator Overloading.

For every operator Magic Methods are available. To overload any operator we have to override that Method in our class.
Internally + operator is implemented by using __add__() method.This method is called magic method for + operator. We have to override this method in our class.

The following is the list of operators and corresponding magic methods.

+ ---> object.__add__(self,other)
- ---> object.__sub__(self,other)
*** ---> object.__mul__(self,other)**
/ ---> object.__div__(self,other)
// ---> object.__floordiv__(self,other)
% ---> object.__mod__(self,other)
**** ---> object.__pow__(self,other)**
+= ---> object.__iadd__(self,other)
-= ---> object.__isub__(self,other)
***= ---> object.__imul__(self,other)**
/= ---> object.__idiv__(self,other)
//= ---> object.__ifloordiv__(self,other)
%= ---> object.__imod__(self,other)
****= ---> object.__ipow__(self,other)**
< ---> object.__lt__(self,other)
<= ---> object.__le__(self,other)
> ---> object.__gt__(self,other)
>= ---> object.__ge__(self,other)
== ---> object.__eq__(self,other)
!= ---> object.__ne__(self,other)

2. Method Overloading:

If 2 methods having same name but different type of arguments then those methods are said to be overloaded methods.

Eg: m1(int a)
m1(double d)

But in Python Method overloading is not possible.
If we are trying to declare multiple methods with same name and different number of arguments then Python will always consider only last method.

How we can handle overloaded method requirements in Python:

Most of the times, if method with variable number of arguments required then we can handle with default arguments or with variable number of argument methods.

3. Constructor Overloading:

Constructor overloading is not possible in Python.
If we define multiple constructors then the last constructor will be considered.

Abstract Method:

Sometimes we don't know about implementation,still we can declare a method. Such type of methods are called abstract methods.i.e abstract method has only declaration but not implementation.
In python we can declare abstract method by using @abstractmethod decorator as follows.

```
@abstractmethod
def m1(self): pass
```

@abstractmethod decorator present in abc module. Hence compulsory we should import abc module,otherwise we will get error.
abc==>abstract base class module

Child classes are responsible to provide implemention for parent class abstract methods.

Abstract class:

Some times implementation of a class is not complete,such type of partially implementation classes are called abstract classes. Every abstract class in Python should be derived from ABC class which is present in abc module.

Conclusion: If a class contains atleast one abstract method and if we are extending ABC class then instantiation is not possible.

"abstract class with abstract method instantiation is not possible"

Parent class abstract methods should be implemented in the child classes. otherwise we cannot instantiate child class.If we are not creating child class object then we won't get any error.

Note: If we are extending abstract class and does not override its abstract method then child class is also abstract and instantiation is not possible.

Note: Abstract class can contain both abstract and non-abstract methods also.

Interfaces In Python:

In general if an abstract class contains only abstract methods such type of abstract class is considered as interface.

Note: The inbuilt function globals()[str] converts the string 'str' into a class name and returns the classname.

Concreate class vs Abstract Class vs Inteface:

1. If we dont know anything about implementation just we have requirement specification then we should go for interface.

2. If we are talking about implementation but not completely then we should go for abstract class.(partially implemented class)

3. If we are talking about implementation completely and ready to provide service then we should go for concrete class.

Public, Protected and Private Attributes:

By default every attribute is public. We can access from anywhere either within the class or from outside of the class.

Protected attributes can be accessed within the class anywhere but from outside of the class only in child classes. We can specify an attribute as protected by prefexing with _ symbol.

syntax:
_variablename=value

But is is just convention and in reality does not exists protected attributes.

private attributes can be accessed only within the class.i.e from outside of the class we cannot access. We can declare a variable as private explicitly by prefexing with 2 underscore symbols.

syntax: __variablename=value

How to access private variables from outside of the class:

We cannot access private variables directly from outside of the class.
But we can access indirectly as follows

objectreference._classname__variablename

__str__() method:

Whenever we are printing any object reference internally __str__() method will be called which is returns string in the following format

<__main__.classname object at 0x022144B0>

To return meaningful string representation we have to override __str__() method.