1.2 Abstraction

An abstract class can have both abstract methods and concrete methods.

We can now access the concrete method of the abstract class by instantiating an object of the child class.

We can also access the abstract methods of the child classes with it.

Interesting points to keep in mind are:

We always need to provide an implementation of the abstract method in the child class even when implementation is given in the abstract class.

A subclass must implement all abstract methods that are defined in the parent class otherwise it results in an error.

In [1]:

```
from abc import ABC,abstractmethod

class Animal(ABC):

    #concrete method
    # inherited
    def sleep(self):
        print("I am going to sleep in a while")


    @abstractmethod
    def sound(self):
        print("This function is for defining the sound by any animal")

#Subclasses or child classes
class Snake(Animal):
    def sound(self):
        print("I can hiss")

class Dog(Animal):
    def sound(self):
        print("I can bark")

class Lion(Animal):
    def sound(self):
        print("I can roar")

class Cat(Animal):
    #overriding
    def sound(self):
        print("I can meow")
```

In [2]:

```
c = Cat()
c.sleep()
c.sound()

c = Snake()
```

```
c.sound()
c.sleep()
```

```
I am going to sleep in a while
I can meow
I can hiss
I am going to sleep in a while
```

In [20]:

```
class Rabbit(Animal):
    def sound(self):
        super().sound()
        print("I can squeak")


c = Rabbit()
c.sound()
```

```
This function is for defining the sound by any animal
I can squeak
```

In [12]:

```
# Error since you can instantiate abstract class with creating a method
class Deer(Animal):

    def sound(self):
        pass

c = Deer()
c.sound()
c.sleep()
```

```
I am going to sleep in a while
```

1.3 Inheritance

1.3.1 Single member

In [2]:

```
# single inheritance example

class parent:
    def func1(self):
        print("Hello Parent")

class child(parent):
    def func2(self):
        print("Hello Child")


test = child()
test.func1()
test.func2()
```

```
Hello Parent
Hello Child
```

1.3.2 Multiple member

In [3]:

```
class parent1:
    def func1(self):
        print("Hello Parent1")

class parent2:
    def func2(self):
        print("Hello Parent2")

class parent3:
    def func2(self):
        print("Hello Parent3")

class child(parent1, parent2, parent3):
    def func3(self):
        print("Hello Child")

# Driver Code
test = child()
test.func1()
test.func2()
test.func3()

print(child.__mro__)
```

```
Hello Parent1
Hello Parent2
Hello Child
(<class '__main__.child'>, <class '__main__.parent1'>, <class '__main__.pare
nt2'>, <class '__main__.parent3'>, <class 'object'>)
```

Here, the first parameter references the child class / subclass the function is used in.

- issubclass(): The issubclass() function is a convenient way to check whether a class is the child of the parent class.

In other words, it checks if the first class is derived from the second class. If the classes share a parent-child relationship, it returns a boolean value of True. Otherwise, False.

- isinstance(): isinstance() is another inbuilt function of python which allows us to check

whether an object is an instance of a particular class or any of the classes it has been derived from. It takes two parameters

i.e. the object and the class we need to check it against. It returns a boolean value of True if the object is an instance and otherwise, False.

Quiz

In [43]:

```
# issubclass() and isinstance() example

class parent:                     # parent class
    def func1(self):
        print("Hello Parent")

class child(parent):              # child class
    def func2(self):
        print("Hello Child")
print(issubclass(child,parent))      # checks if child is subclass of parent

print(issubclass(parent,child))         # checks if parent is subclass of ch

A = child()                         # objects initialized
B = parent()

print(isinstance(A,child))                # checks if A is instance of chilc
print(isinstance(A,parent))               # checks if A is instance of paren
print(isinstance(B,child))                # checks if B is instance of chilc
print(isinstance(B,parent))               # checks if B is instance of paren
```

```
True
False
True
True
False
True
```

1.4 Polymorphism

In [1]:

```
from math import pi
class Shape:

    def __init__(self, name):
        self.name = name

    def area(self):
        pass

    def fact(self):
        return "I am a two-dimensional shape."

    def __str__(self):
        return self.name



class Circle(Shape):

    def __init__(self, radius):
        super().__init__("Circle")
        self.radius = radius

    def area(self):
        return pi*self.radius**2

shape_circle = Circle(7)

print(shape_circle)
print(shape_circle.area())
print(shape_circle.fact())
```

```
Circle
153.93804002589985
I am a two-dimensional shape.
```

Main concept of Polymorphism is method overriding

There are certain prerequisites for method overriding in Python. They're discussed below --

- Method overriding cannot be done within a class. So,we need to derive a child class from a parent class. Hence Inheritance is mandatory.
- The method must have the same name as in the parent class
- The method must have the same number of parameters as in the parent class.