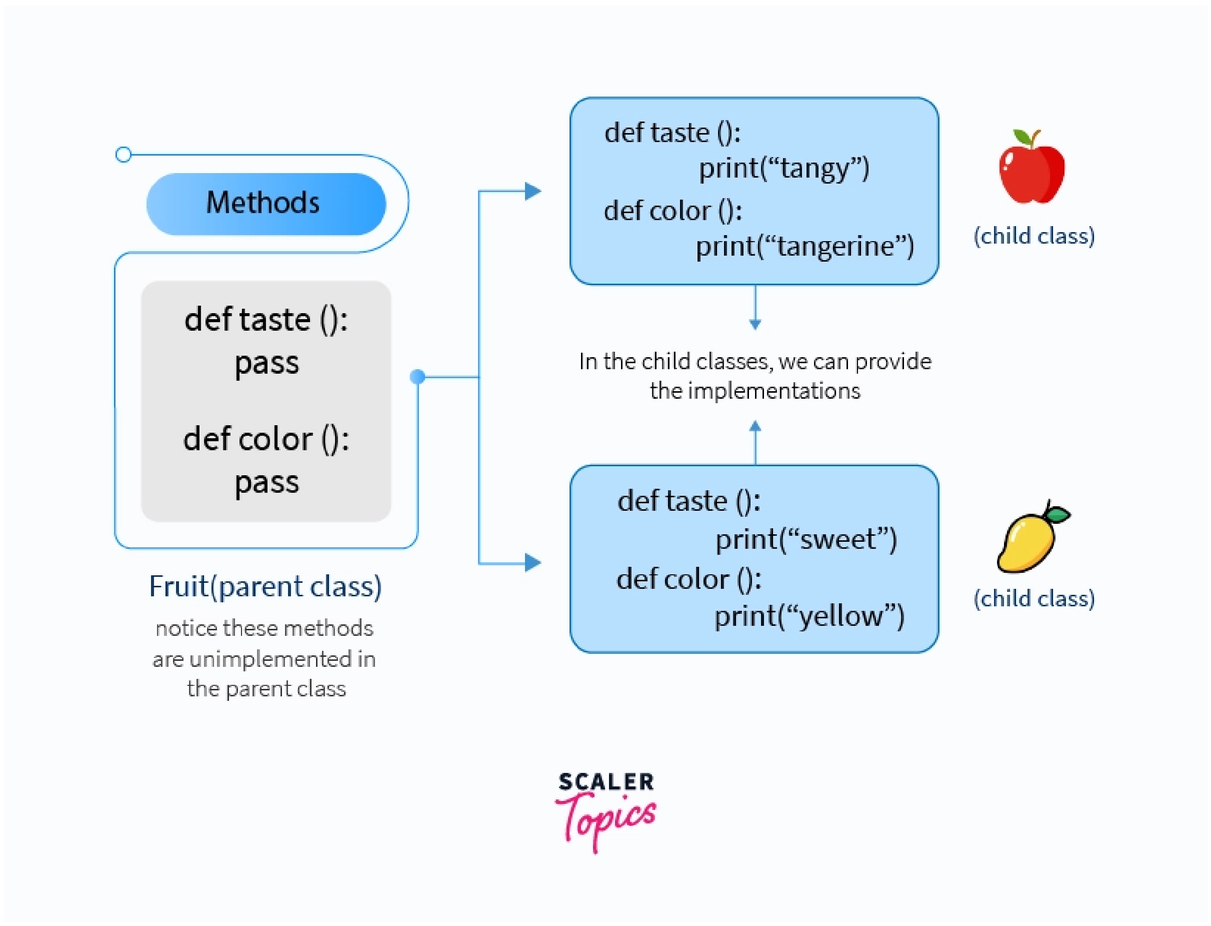
Abstract Classes In Python

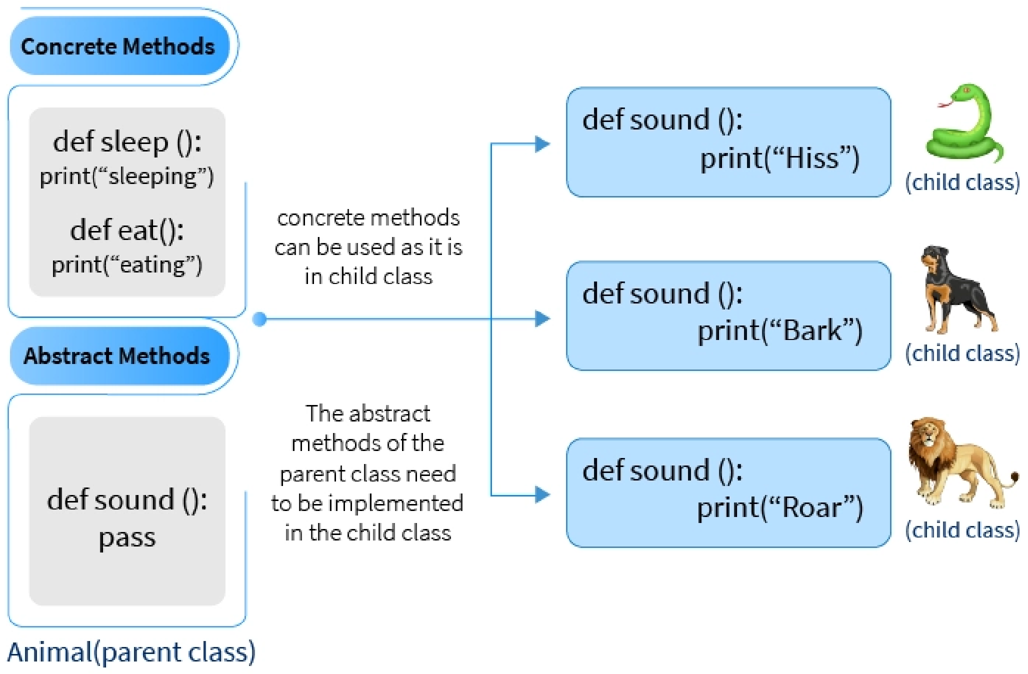
An abstract class can be considered as a **blueprint for other classes**. It **allows you to create a set of methods that must be created within any child classes built from the abstract class.**

A class which contains one or more abstract methods is called an abstract class.

An abstract method is a method that has a declaration but does not have an implementation.

While we are designing large functional units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class.


Methods
def taste ():
pass
def color ():
pass
Fruit(parent class)
notice these methods are unimplemented in the parent class
def taste ():
print(“tangy”)
def color ():
print(“tangerine”)
(child class)
In the child classes, we can provide the implementations
def taste ():
print(“sweet”)
def color ():
print(“yellow”)
(child class)
SCALER Topics


Concrete Methods
def sleep ():
print("sleeping")
def eat():
print("eating")
Abstract Methods
def sound ():
pass
Animal(parent class)
concrete methods can be used as it is in child class
The abstract methods of the parent class need to be implemented in the child class
def sound ():
print("Hiss")
(child class)
def sound ():
print("Bark")
(child class)
def sound ():
print("Roar")
(child class)

```python
# Python program showing
# abstract base class work

from abc import ABC, abstractmethod

class Polygon(ABC):

    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")
```

```
# Driver code
R = Triangle()
R.noofsides()

K = Quadrilateral()
K.noofsides()

R = Pentagon()
R.noofsides()

K = Hexagon()
K.noofsides()
```

```python
# Python program showing
# abstract base class work

from abc import ABC, abstractmethod
class Animal(ABC):

    def move(self):
        pass

class Human(Animal):

    def move(self):
        print("I can walk and run")

class Snake(Animal):

    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")

# Driver code
R = Human()
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

```
# Python program showing
# abstract properties

import abc
from abc import ABC, abstractmethod

class parent(ABC):
   @abc.abstractproperty
   def geeks(self):
      return "parent class"
class child(parent):

   @property
   def geeks(self):
      return "child class"


try:
   r =parent()
   print( r.geeks)
except Exception as err:
   print (err)

r = child()
print (r.geeks)
```

```
# Python program showing
# abstract class cannot
# be an instantiation
from abc import ABC,abstractmethod

class Animal(ABC):
   @abstractmethod
   def move(self):
      pass
class Human(Animal):
   def move(self):
      print("I can walk and run")

class Snake(Animal):
   def move(self):
      print("I can crawl")

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

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

c=Animal()
```

```python
from abc import ABC,abstractmethod

class Animal(ABC):

    #concrete method
    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")
        pass

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):
    def sound(self):
        print("I can meow")


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

c = Snake()
c.sound()
```