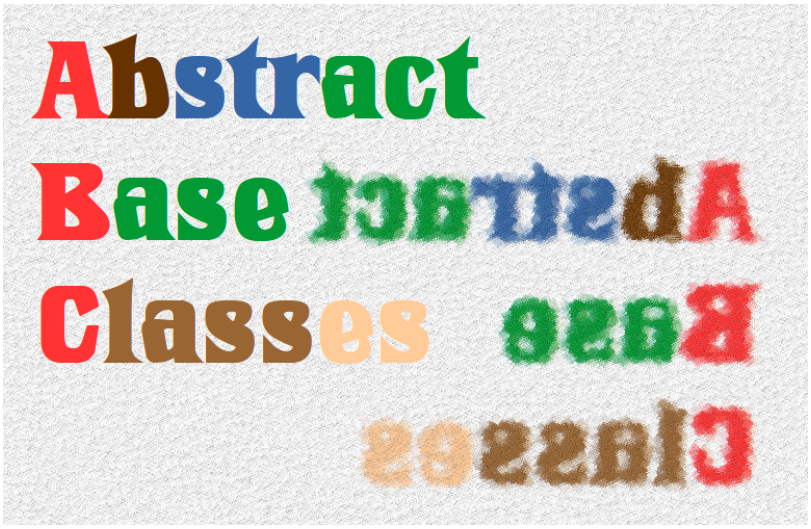
# ABSTRACT CLASSES

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes cannot be instantiated, and require subclasses to provide implementations for the abstract methods.




You can see this in the following examples:

```
class AbstractClass:

    def do_something(self):
        pass


class B(AbstractClass):
    pass

a = AbstractClass()
b = B()
```

If we start this program, we see that this is not an abstract class, because:

- we can instantiate an instance from
- we are not required to implement do_something in the class defintition of B

Our example implemented a case of simple inheritance which has nothing to do with an abstract class. In fact, Python on its own doesn't provide abstract classes. Yet, Python comes with a module which provides the infrastructure for defining Abstract Base Classes (ABCs). This module is called - for obvious reasons - abc.

The following Python code uses the abc module and defines an abstract base class:

```
from abc import ABC, abstractmethod

class AbstractClassExample(ABC):

    def __init__(self, value):
        self.value = value
        super().__init__()

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

We will define now a subclass using the previously defined abstract class. You will notice that we haven't implemented the do_something method, even though we are required to implement it, because this method is decorated as an abstract method with the decorator "abstractmethod". We get an exception that DoAdd42 can't be instantiated:

```
class DoAdd42(AbstractClassExample):
    pass

x = DoAdd42(4)
```

```
---------------------------------------------------------------
---------------
TypeError                                 Traceback (most r
ecent call last)
<ipython-input-4-2bcc42ab0b46> in <module>
      2     pass
      3
----> 4 x = DoAdd42(4)

TypeError: Can't instantiate abstract class DoAdd42 with ab
stract methods do_something
```

We will do it the correct way in the following example, in which we define two classes inheriting from our abstract class:

```
class DoAdd42(AbstractClassExample):

    def do_something(self):
        return self.value + 42

class DoMul42(AbstractClassExample):

    def do_something(self):
        return self.value * 42

x = DoAdd42(10)
y = DoMul42(10)

print(x.do_something())
print(y.do_something())
```

```
52
420
```

A class that is derived from an abstract class cannot be instantiated unless all of its abstract methods are overridden.

You may think that abstract methods can't be implemented in the abstract base class. This impression is wrong: An abstract method can have an implementation in the abstract class! Even if they are implemented, designers of subclasses will be forced to override the implementation. Like in other cases of "normal" inheritance, the abstract method can be invoked with super() call mechanism. This enables providing some basic functionality in the abstract method, which can be enriched by the subclass implementation.

```
from abc import ABC, abstractmethod

class AbstractClassExample(ABC):

    @abstractmethod
    def do_something(self):
        print("Some implementation!")

class AnotherSubclass(AbstractClassExample):

    def do_something(self):
        super().do_something()
        print("The enrichment from AnotherSubclass")

x = AnotherSubclass()
x.do_something()
```

```
Some implementation!
The enrichment from AnotherSubclass
```