

Programação Orientada a Objetos

Classes-base Abstratas

Abstract Base Class (ABCMeta) e abstractmethod

Classes-base Abstratas

Python suporta classes-base abstratas (*Abstract Base Classes*), que são metaclasses que permitem forçar a implementação de determinados métodos e atributos das classes e subclasses derivadas.

O módulo *abc* define a metaclasses *ABCMeta* e o decorador *abstractmethod* que identifica o método que deve ser implementado.

Combinando *abstractmethod* com *property*, podemos ter propriedades abstratas.

Classes-base Abstratas

Um método abstrato é um método que é declarado, mas não contém implementação. As classes abstratas não podem ser instanciadas e exigem subclasses para fornecer implementações para os métodos abstratos.

Uma subclasse de uma classe abstrata só pode ser instanciada se todos os métodos abstratos estiverem sido implementados.

Métodos abstratos podem ser implementados em uma classe abstrata, porém, estes serão sobrescritos nas classes derivadas.

Como em casos de herança normal, os métodos da classe abstrata podem ser invocados usando o *super()*.

Classes-base Abstratas

Exemplo 1 – Método abstrato implementado:

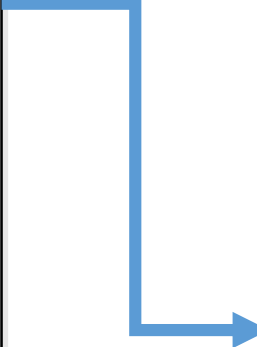
```
from abc import ABCMeta, abstractmethod

class Carro(metaclass=ABCMeta):
    @abstractmethod
    def ligar(self):
        # Sem implementação
        pass

class fusca(Carro):
    def __init__(self):
        print("Vou implementar o ligar.")

    def ligar(self):
        print("Uhu, o fusca ligou!!!")

fusquinha = fusca()
fusquinha.ligar()
```



Vou implementar o ligar.
Uhu, o fusca ligou!!!

Classes-base Abstratas

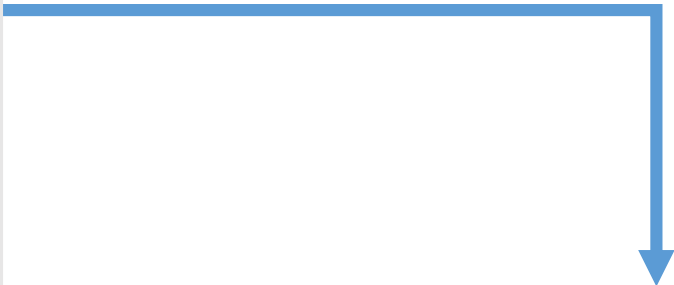
Exemplo 2 – Método abstrato não implementado:

```
from abc import ABCMeta, abstractmethod

class Carro(metaclass=ABCMeta):
    @abstractmethod
    def ligar(self):
        # Sem implementação
        pass

class kombi(Carro):
    def __init__(self):
        print("Não vou implementar o ligar.")

furgao = kombi()
```



Traceback (most recent call last):
File ".../exemplo1.py", line 13, in <module>
 furgao = kombi()
TypeError: Can't instantiate abstract class kombi with abstract methods ligar

Classes-base Abstratas

Exemplo 3 – Propriedade abstrata implementada:

```
from abc import ABCMeta, abstractmethod

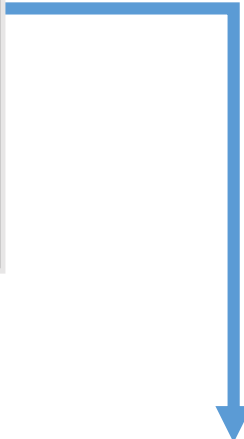
class Carro(metaclass=ABCMeta):
    @property
    @abstractmethod
    def valor(self):
        return "Retorno de valor"

    @abstractmethod
    def ligar(self):
        # Sem implementação
        pass
```

```
class fusca(Carro):
    @property
    def valor(self):
        return "Propriedade concreta."

    def ligar(self):
        print("Uhu, o fusca ligou!!!")

fusquinha = fusca()
fusquinha.ligar()
print(fusquinha.valor)
```



Uhu, o fusca ligou!!!
Propriedade concreta.

Classes-base Abstratas

Exemplo 4 – Propriedade abstrata não implementada:

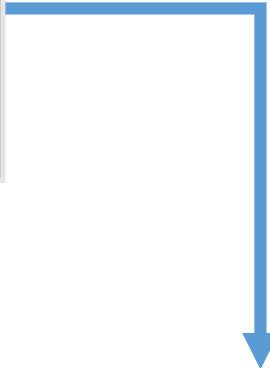
```
from abc import ABCMeta, abstractmethod

class Carro(metaclass=ABCMeta):
    @property
    @abstractmethod
    def valor(self):
        return "Retorno de valor"

    @abstractmethod
    def ligar(self):
        # Sem implementação
        pass
```

```
class fusca(Carro):
    def ligar(self):
        print("Uhu, o fusca ligou!!!")

fusquinha = fusca()
fusquinha.ligar()
print(fusquinha.valor)
```



Traceback (most recent call last):
File ".../exemplo4.py", line 18, in <module>
 fusquinha = fusca()
TypeError: Can't instantiate abstract class fusca with abstract methods valor

FIM