Metaclasses

terça-feira, 8 de novembro de 2022 19:21

```python
# Quando criamos uma classe, a classe serve como um molde para criação de um objeto.
# Tudo em Python é um objeto - Classes por exemplo também são Objetos.
# O objeto é a instância que recebe a classe.

#classe A criada
class A:
    attr = "Valor"

#Instâncias são iguais à Objetos
#objeto1 = instância1 etc...
objeto1 = A()
objeto2 = A()
objeto3 = A()
```

```python
# Para criar uma metaclasse devemos criar usando o type como herança dessa classe.

class Meta(type):
    #mcs = metaclasse, name = nome da classe que esta sendo criada, bases = classes Pais dessa classe, namespace = espaço onde cria atributos da classe.
    def __new__(mcs, name, bases, namespace):
        #mandando tudo isso para a classe Meta
        if name == "Pai":  # Nao quero modificar o comportamento da classe Pai.
            return type.__new__(mcs, name, bases, namespace)


        return type.__new__(mcs, name, bases, namespace)
```

```python
class Pai(metaclass = Meta):  # usando a metaclasse Meta na classe Pai, toda classe que for criada ou herdada por essa classe Pai
                              # automaticamente vai passar na Metaclass
    def falar(self):
        self.b_falar()


#Herdando da classe Pai
class Filha(Pai):
    pass
    #def b_falar(self):
    #    print("oi")

#instânciando um objeto a classe filha
objeto_filha = Filha()
#chamando um método da classe Pai
objeto_filha.falar()  #Chamando o metodo da classe Pai que recebe um método da classe filha printando oi

# citando um exemplo, a class Pai é uma classe que faz as coisas da parte de biblioteca e a Classe filha é responsável por criar a parte da interface grafica
# Quando não existe o metodo (b_falar) da classe Filha que é solicitado na classe Pai (metodo falar()) encontramos um erro. Ex: supondo
# que o metodo b_falar não fosse criado.

# Um jeito de resolver esse problema seria criando uma classe abstrata que chamasse o metodo b_falar(), porém existe outro metodo
```

```python
# citando um exemplo, a class Pai é uma classe que faz as coisas da parte de biblioteca e a Classe filha é responsável por criar a parte da interface grafica
# Quando não existe o metodo (b_falar) da classe Filha que é solicitado na classe Pai (metodo falar()) encontramos um erro. Ex: supondo
# que o metodo b_falar não fosse criado.

# Um jeito de resolver esse problema seria criando uma classe abstrata que chamasse o metodo b_falar(), porém existe outro metodo



#-----------------------------------------------------------------------------------------------------------------------------------------------------
# Para criar uma metaclasse devemos criar usando o type como herança dessa classe.

class Meta(type):
    #mcs = metaclasse, name = nome da classe que esta sendo criada, bases = classes Pais dessa classe, namespace = espaço onde cria atributos da classe.
    def __new__(mcs, name, bases, namespace):
        #mandando tudo isso para a classe Meta
        if name == "Pai":  # Nao quero modificar o comportamento da classe Pai.
            return type.__new__(mcs, name, bases, namespace)
        #Queremos então verificar na classe filha se existe o metodo b_fala() para o metodo falar funcionar.
        #namespace = tudo que está dentro da classe ex: metodo,variavel
        if 'b_fala' not in namespace:
            print("Oi,Você precisa criar o metodo b_fala()")
        else:
            if not callable(namespace['b_fala']):  # varificando se existe um metodo com o nome b_fala dentro da classe restante no caso Filha.
                print("b_fala precisa ser um método.")

        return type.__new__(mcs, name, bases, namespace)
```

```python
class Pai(metaclass = Meta):  # usando a metaclasse Meta na classe Pai, toda classe que for criada ou herdada por essa classe Pai
                              # automaticamente vai passar na Metaclass
    def falar(self):
        self.b_falar()


#Herdando da classe Pai
class Filha(Pai):
    def b_falar(self):
        print("oi")
```

```python
#Outros exemplos: #----------------------------------------------------------------------------------------------------------------------------------

class Meta(type):
    #mcs = metaclasse, name = nome da classe que esta sendo criada, bases = classes Pais dessa classe, namespace = espaço onde cria atributos da classe.
    def __new__(mcs, name, bases, namespace):
        #mandando tudo isso para a classe Meta
        if name == "Pai":  # Nao quero modificar o comportamento da classe Pai.
            return type.__new__(mcs, name, bases, namespace)
        if 'atributo' in namespace:  # verificando se tem o atributo nas classes que não sejam a classe Pai
            del namespace['atributo']  # Excluindo as sobrescrições

        return type.__new__(mcs, name, bases, namespace)


class Pai(metaclass = Meta):  # usando a metaclasse Meta na classe Pai, toda classe que for criada ou herdada por essa classe Pai
                              # automaticamente vai passar na Metaclass
    atributo = "valor A"  # não quero que esse atributo seja sobrescrito logo eu modifico na metaclasse para que isso não aconteça;

#Herdando da classe Pai
class Filha(Pai):
    atributo = "Valor B"

b = Filha()
print(b.atributo)  # o valor de atributo que deveria ser puxado da classe que está como herança(classe Pai) seria valor A,
# porém a classe Filha está sobrescrevendo o valor de atributo
```

```python
#Criando classes com type: #----------------------------------------------------------------------------------------------------------------------------------

A = type(" nome da classe", ('herando de classe'), {'Atributo': 'Olá Mundo!'}  )  # ou seja a função type cria classes.

a = A()
print(a.atributo)

# Exemplo com outra classe herdando.

class B:
    nome = 'geronimo'

A = type(" nome da classe", (B), {'Atributo': 'Olá Mundo!'}  )  # ou seja a função type cria classes.

a = A()
print(a.nome)  # imprime o valor de nome.
```