100 XP

Usar classes e variáveis para transferir o modelo OOP para o código

6 minutos

É interessante avaliar a OOP (programação orientada a objeto) criando o jogo pedra, papel e tesoura. Para fazer isso, primeiro você precisa modelá-lo em um formato OOP.

Precisaremos aplicar alguns conceitos básicos de OOP, como classes, objetos e estado. Esta unidade explora as seguintes partes:

- **Conceitos de OOP importantes**: para analisar nos termos da OOP, você precisa entender alguns conceitos básicos como classes, objetos e estado. Você precisa saber qual é a diferença entre eles e como eles se relacionam uns com os outros.
- **Variáveis na OOP**: você precisa saber como lidar com variáveis e como adicioná-las aos seus objetos.

O que é um objeto?

O conceito de objetos foi mencionado algumas vezes já como parte da tentativa de *modelar* domínios do problema. Um objeto é um ator. É algo que faz alguma coisa dentro de um sistema. Em decorrência da execução de uma ação, ela altera o estado dentro de si mesma ou em outros objetos.

Vamos imaginar um objeto no mundo real. Você está em um estacionamento. O que você vê? É provável que você veja muitos carros, de diferentes formas, tamanhos e cores. Para descrever um carro, você pode usar propriedades como marca, modelo, cor e tipo de carro. Se você atribui valores a essas propriedades, fica claro rapidamente se você está falando sobre um Ferrari vermelho, Jeep com tração nas quatro rodas, um Mustang amarelo e assim por diante.

Em outra cena, imagem de um baralho em Las Vegas. Você examina dois cartões diferentes, que são dois objetos. Você percebe que eles têm algumas propriedades comuns, a saber, naipes. O naipe dos objetos pode ser paus, copas, ouros ou espadas. Os valores deles podem ser ás, rei, nove e assim por diante.

Dica

Olhe ao seu redor. Pegue dois objetos semelhantes, como dois livros ou duas cadeiras, e veja se é possível encontrar as propriedades que as descrevem melhor e as diferenciam.

O que é uma classe?

Uma classe é um tipo de dados que funciona como uma definição de modelo para um determinado tipo de objeto.

Você aprendeu que um sistema OOP tem objetos, e esses objetos têm propriedades. Há um conceito semelhante ao de objeto, que é o conceito de classe.

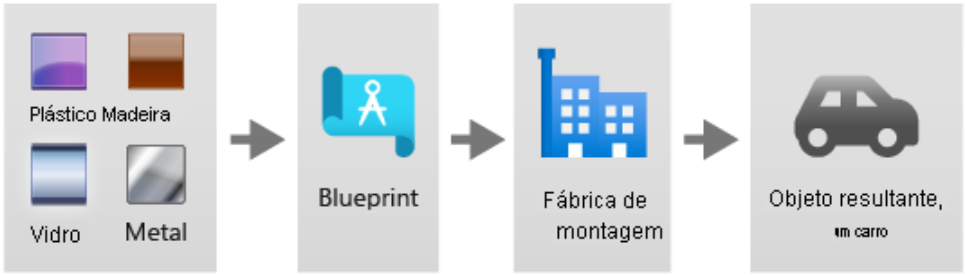
Então, o que é uma classe? Uma classe é um blueprint de um objeto. Se a classe é o blueprint de um carro, o objeto é o carro que você dirige. A classe é o que você escreve no código. O objeto é o que você obtém quando solicita que o ambiente de runtime execute seu código.

Esta é uma tabela de alguns exemplos de classes e os objetos resultantes:

Expandir a tabela

Classe	Objeto
Blueprint de um carro	Honda Accord, Jeep Wrangler
Gato	Garfield, o gato
Descrição de sorvete	Morango, chocolate ou baunilha

A maneira como você cria um objeto com base em uma classe é semelhante à maneira como você criaria um carro com base em um blueprint. Quando você cria um objeto, seu programa solicita ao sistema operacional por recursos, como memória, para poder construir o objeto. Por outro lado, quando um carro é feito com base em um blueprint, a fábrica solicita recursos como metal, borracha e vidro para poder montar o carro.



Criar uma classe

Uma classe em Python é criada usando a palavra-chave `class` e dando a ela um nome, como neste exemplo:

Python

```
class Car:
```

Criar um objeto com base em uma classe

Quando você cria um objeto com base em uma classe, dizemos que você o *instancia*. O que você faz é pedir ao sistema operacional que, com esse modelo (a classe) e esses valores iniciais, ele forneça memória suficiente e crie um objeto. Essencialmente, instanciar é outra palavra para criar.

Para instanciar um objeto, você adiciona parênteses ao nome da classe. O que você obtém é um objeto que você pode optar por atribuir a uma variável, dessa forma:

Python

```
car = Car()
```

`car` é a variável que contém a instância do objeto. O momento de instanciação, ou criação do objeto, é quando você chama `Car()`.

Variáveis na OOP versus variáveis em programas procedurais

Com base na programação procedural, você está acostumado a ter variáveis para conter as informações e acompanhar o estado. Você pode definir essas variáveis sempre que precisar delas em seu arquivo. A *inicialização* de uma variável típica pode ter esta aparência:

Python

```
pi = 3.14
```

Você também tem variáveis na OOP, embora elas sejam *anexadas* a objetos em vez de serem definidas por conta própria. Você faz referência a variáveis em um objeto como *atributos*. Quando um atributo é anexado a um objeto, ele é usado de uma das duas maneiras:

- **Descrever o objeto**: um exemplo de uma variável de *descrição* é, por exemplo, a cor de um carro ou o número de manchas em uma girafa.
- **Conter o estado**: uma variável pode ser usada para descrever o estado de um objeto. Um exemplo de estado é o andar de um elevador ou se ele está em operação ou não.

Adicionar atributos a uma classe

Saber quais atributos (variáveis) devem ser adicionados à sua classe faz parte da modelagem. Você aprendeu a criar uma classe no código. Então como adicionar um atributo a ela? Você precisa informar à

classe quais atributos ela deve ter no momento da construção, quando um objeto está sendo instanciado. Há uma função especial que está sendo chamada no momento da criação, chamada de *construtor*.

Construtor

Muitas linguagens de programa têm a noção de um construtor, a função especial que só é invocada quando o objeto é criado pela primeira vez. O construtor será chamado apenas uma vez. Nesse método, você cria os atributos que o objeto deve ter. Além disso, você atribui valores iniciais aos atributos criados.

Em Python, o construtor tem o nome `__init__()`. Você também precisa passar um parâmetro especial `self` para o construtor. O parâmetro `self` refere-se à instância do objeto. Qualquer atribuição a essa palavra-chave significa que o atributo termina na instância do objeto. Se você não adicionar um atributo a `self`, ele será tratado como uma variável temporária que não existirá depois que a execução de `__init__()` for concluída.

> **Observação**
>
> O parâmetro `self` também precisará ser passada para métodos que precisem se referir a qualquer coisa na instância do objeto. Esse conceito será abordado na próxima unidade.

Adicionar e inicializar atributos em uma classe

Vejamos um exemplo de configuração de atributos em um construtor:

Python

```
class Elevator:
    def __init__(self, starting_floor):
        self.make = "The elevator company"
        self.floor = starting_floor

# To create the object

elevator = Elevator(1)
print(elevator.make) # "The Elevator company"
print(elevator.floor) # 1
```

O exemplo anterior descreve a classe `Elevator` com duas variáveis, `make` e `floor`. Uma importante vantagem do código é que `__init__()` é chamado implicitamente. Você não chama o método `__init__()` por nome, mas ele é chamado quando o objeto é criado, nesta linha de código:

Python

```
elevator = Elevator(1)
```

Uso incorreto de `self`

Para enfatizar como o parâmetro `self` funciona, considere o seguinte código no qual dois atributos, `color` e `make`, estão sendo atribuídos no construtor `__init__()`:

Python

```
class Car:
    def __init__():
        self.color = "Red" # ends up on the object
        make = "Mercedes" # becomes a local variable in the constructor

car = Car()
print(car.color) # "Red"
print(car.make) # would result in an error, `make` does not exist on the object
```

Se a finalidade fosse tornar `color` e `make` os atributos da classe `Car`, você precisaria modificar o código. No construtor, verifique se ambos os atributos são atribuídos a `self`, desta forma:

Python

```
self.color = "Red" # ends up on the object
self.make = "Mercedes"
```

Dica

Como lição de casa, tente criar uma classe com base em um livro que você leria. Pense em como você escreveria a classe e quais propriedades ela deveria ter.

Unidade seguinte: Exercício – Modelar e realizar scaffold do seu jogo

Continuar >