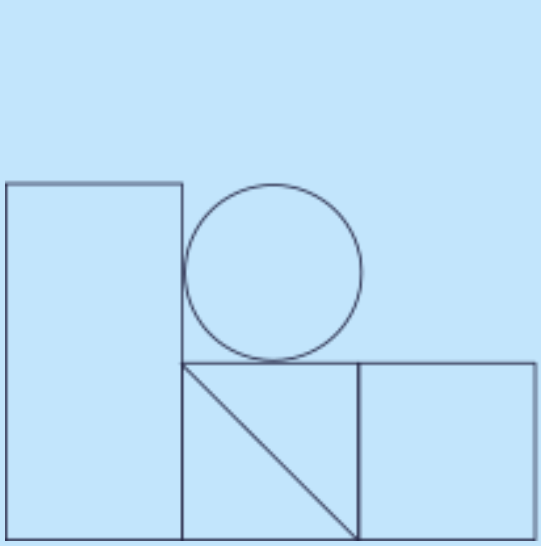
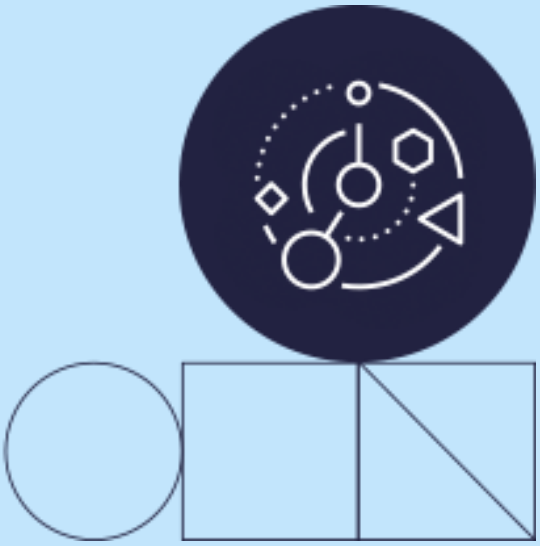
IBM SkillsBuild

IBM

OOP com Python

Objetos e classes





Índice

Introdução

Como vimos no tópico anterior, no OOP sempre trabalhamos com objetos e todos os nossos objetos terão atributos e métodos.

O primeiro e mais importante conceito no OOP é a distinção entre classe e objeto.

Uma **classe** é um modelo. Define de forma genérica como serão os objetos de um determinado tipo. Por exemplo, uma classe para representar animais pode ser chamada 'animal' e ter uma série de atributos, como 'nome' ou 'idade' (que geralmente são propriedades), e uma série de comportamentos que eles podem ter, como caminhar ou comer, e que por sua vez são implementados como métodos da classe (funções).

Objetos

O que é um objeto

Um simples exemplo de um objeto, como dissemos antes, poderia ser um determinado animal, por exemplo, um cão. Um cão tem uma idade, então criamos um novo atributo de "idade" e, além disso, ele pode envelhecer, por isso definimos um novo método. Dados e lógica. Isto é o que é definido em muitos programas como a definição de uma classe, que é a definição global e genérica de muitos objetos.

Os atributos determinarão as qualidades de nosso objeto e os métodos as funcionalidades. Por exemplo, se quiséssemos criar objetos do tipo carro, nossos objetos teriam atributos; marca, modelo, cor, peso...etc. e métodos; partida, freio, volta...etc.

Objeto	Propiedades	Métodos
	car.marca = Seat car.modelo = Leon car.peso = 950kg car.color = Rojo	car.arrancar () car.girar () car.frenar () car.acelerar ()

Quando convertermos isto em código, veremos que os atributos serão sempre variáveis (números, strings...etc) e os métodos devem ser funções.

O processo de "pensar" em nosso código como um conjunto de objetos é algo que, no início, o estudante pode achar bastante difícil, mas em Python já temos usado objetos constantemente sem sabê-lo. Por exemplo, a janela de execução de um programa é um objeto, e como objeto, ele tem atributos; uma largura, uma altura, uma cor, uma aparência...etc. E tem funções; pode ser aberto, fechado, ampliado, pode mostrar informações, etc.

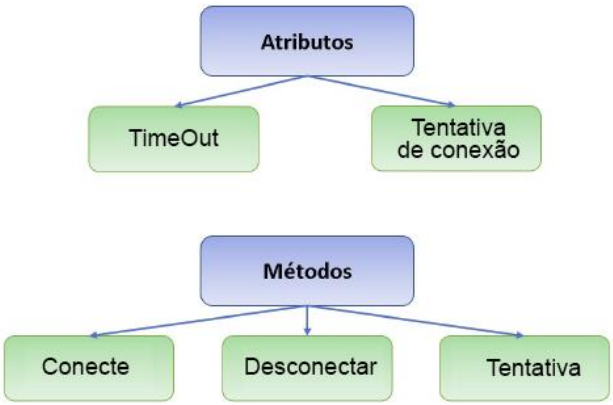
Um usuário de nosso programa também seria um objeto. Seus atributos poderiam ser: ter um login, um password, privilégios... etc. E seus métodos; para poder registrar, cancelar a inscrição, alterar seus dados...etc.

Assim, se pensarmos nisso, podemos deduzir as propriedades e métodos de todos os elementos de um programa; botões, menus... etc.

Um objeto do tipo janela teria os seguintes atributos e métodos (entre muitos outros):

Um objeto do tipo usuário poderia ter os seguintes atributos e métodos:

Podemos até encontrar objetos cujos atributos e métodos podem parecer mais difíceis de determinar a princípio, tais como uma conexão de banco de dados:



Como podemos ver, o paradigma OOP nos fará ver nosso programa como um conjunto de objetos que se relacionam uns com os outros. No próximo capítulo veremos um conceito estreitamente relacionado ao de objeto, o conceito de classe, e como codificar nossas classes e objetos.

Abstração

Estreitamente relacionado com os conceitos de classe e objeto está o conceito de abstração.

Temos visto vários exemplos de objetos como o objeto janela, usuário, conexão... etc. Estes objetos têm atributos e métodos que utilizaremos em nossos programas. De fato, qualquer objeto que criarmos em nosso programa terá atributos e métodos e, estritamente falando, um número quase ilimitado de atributos e métodos poderia ser retirado de quase qualquer objeto. Um carro, por exemplo, poderia ter como atributos: altura, comprimento, largura, peso, cor, marca, modelo, placa numérica, tipo, ano, país de origem, se for novo ou de segunda mão, se for uma frota privada ou de uma empresa...etc. Por pura lógica, não vamos instanciar todas as possibilidades, mas simplesmente aquelas que nos interessam.

O conceito de abstração consiste em saber escolher somente os atributos e métodos que nos interessam para nosso programa e descartar o resto. A definição estrita de abstração seria "determinar as características específicas de um objeto, aquelas que o distinguem de outros tipos de objetos e que conseguem definir limites conceituais com respeito a quem está fazendo a abstração do objeto".

Suponha que tenhamos que implementar um programa para uma oficina de reparação de automóveis e outro para uma companhia de seguros. Em ambos os casos, teremos que instanciar objetos de classe Carro, mas no caso da garagem estaremos interessados em alguns dados sobre o carro e no caso da seguradora, dados diferentes. Por exemplo, a seguradora pode estar interessada em saber se a apólice de seguro automóvel deve ser paga mensal ou anualmente. O proprietário da garagem pouco se importa com estas informações.

Portanto, uma das ações que devemos tomar antes de começarmos a programar é ser claro sobre os dados que vamos precisar introduzir em nossos objetos. Quais serão os atributos e métodos com os quais precisaremos trabalhar.

Classe

Até agora, vimos o conceito de um objeto e vimos vários exemplos de objetos que podemos incorporar em nossos programas. Mas e se precisarmos criar dezenas ou centenas de objetos do tipo **carro**?. Todos esses carros precisam ser programados um a um, mesmo que tenham propriedades e métodos semelhantes? É aqui que entra em jogo o conceito de classe.

Uma classe é um modelo onde as características comuns a um grupo de objetos são escritas. Uma classe representa um conjunto de objetos que compartilham uma estrutura e um comportamento comuns.

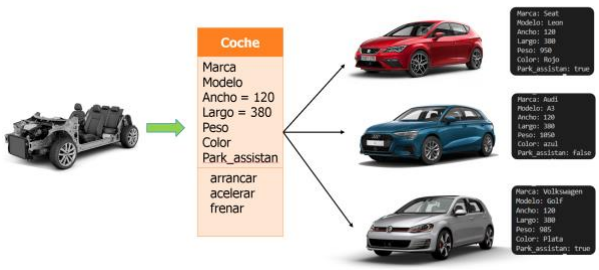
Em outras palavras, uma classe nada mais é do que um **molde para fazer objetos**, um conjunto de especificações que determinam como certos tipos de elementos irão se comportar. Estes elementos, criados a partir das especificações de classe, são o que chamamos de objetos.

Assim, ao nosso modelo OOP acrescentamos um novo elemento, as classes:

Seguindo o exemplo que vimos anteriormente de nosso **carro**, talvez precisemos criar mais carros com exatamente os mesmos atributos, ou similares, ou mesmo totalmente diferentes, mas ainda assim considerados como carros. Então podemos dizer que nosso objeto **carro** pode ser classificado, ou seja, nosso objeto pertence a uma classe, ou seja, à classe **Carro**. Dizemos então que nosso **carro** é uma instância, objeto ou instância da classe **Carro**.

Podemos criar uma classe **Carro**, um molde para fazer carros, a este molde damos as características comuns a todos os objetos da classe do carro e com este molde podemos criar objetos da classe do carro.

Podemos decidir se nossos objetos da classe do carro (nossas instâncias de carro) serão todos iguais ou terão algumas propriedades comuns e algumas específicas do carro.



Neste exemplo, criamos uma classe **Carro** que servirá como modelo para criar instâncias da classe do carro. Criamos a classe **Carro** com uma largura e um comprimento já determinados, então as instâncias de **Carro** terão todos a mesma largura e comprimento, mas o resto das propriedades não estão definidas na classe, teremos que passá-las ao criar cada objeto em particular.

Normalmente, uma aplicação é composta por várias classes, cada uma responsável por um trabalho, que estão relacionadas entre si e com as quais serão criados os objetos necessários ao nosso programa.

Agora vamos levar tudo isso para o código. Para criar uma classe, a sintaxe em Python é:

```
class NombreClase:
```

Por convenção, os nomes das classes começam com letras maiúsculas. Se não o colocarmos em letras maiúsculas, não teremos nenhuma falha, é simplesmente um código de boas práticas de programação.

Dentro da classe, indentados, estariam os atributos e métodos de nossa classe. Lembre-se que os atributos serão variáveis (de fato, eles também são chamados de variáveis de classe) e os métodos são funções.

Continuando com o exemplo de nossa classe de carros, a implementação seria a seguinte:

```
# Declaración de la clase
class Coche():

# Declaración de atributos
    largo = 250
    ancho = 120
    ruedas = 4
    peso = 900
    color = "rojo"
    is_enMarcha = False

# Declaración de métodos
    def arrancar(self):                 # self
hace referencia a la instancia de clase.
        self.is_enMarcha = True         # Es
como si pusiésemos miCoche.is_enMarcha = True

    def estado(self):
        if (self.is_enMarcha == True):
            return "El coche está
arrancado"
        else:
            return "El coche está
parado"
```

Ao criar um método de classe, é obrigatório definir o primeiro parâmetro para ***self***, é como o ***this*** de outros idiomas do OOP. É uma referência ao objeto instanciado na classe.

O uso de ***self*** é necessário porque é uma referência (um ponteiro) para o objeto instanciado. Esta referência, pois indica que um método ou um atributo apontado por ***self*** pertence a uma única instância e não a todos os objetos instanciados da mesma classe.

Referindo-se explicitamente a **`self.is_enMarcha`**, estamos ajudando o intérprete a localizar a referência que queremos que ele encontre. De fato, se não fizermos referência usando **`self.is_enMarcha`** o intérprete procurará diretamente por **`is_enMarcha`** fora da sala de aula, portanto, se **`is_enMarcha`** não existia em nível de módulo, a busca retornaria um erro, já que a variável não foi encontrada.

Criamos o carro de classe e, dentro dele, declaramos:

- Uma série de atributos, que nada mais são do que variáveis:
  - Comprimento, largura, rodas e peso, do tipo number.
  - Color, do tipo string.
  - Is_enMarcha, do tipo boolean.

- Uma série de métodos. Que são funções:
  - Início()
  - Estado()

Já temos nossa classe **carro** criada. Todas as instâncias da classe **carro** que criamos terá os atributos e métodos que colocamos na classe. Mais tarde poderemos mudar seu valor, mas, em princípio, é isso que eles terão.

Agora só precisamos criar tantas cópias de classe (instâncias de classe) quantas quisermos. Vamos criar dois. A sintaxe seria a seguinte:

```
# Declaración de una instancia de clase,
objeto de clase o ejemplar de clase.

miCoche = Coche()
miCoche2 = Coche()
```

Criamos dois objetos chamados **miCoche** e **miCoche2** que são duas instâncias da classe **Carro**, portanto, têm as propriedades e métodos pertencentes à classe **carro**. Agora podemos usar os atributos e métodos da classe do carro nos exemplos que criamos, usando a nomenclatura de pontos, ou seja: **nombreObjeto.atributo** ou **nombreObjeto.propiedad**:

```
# Acceso a un atributo de la clase Coche.
Nomenclatura del punto.
print("El largo del coche es de" ,
miCoche.largo, "cm.")
miCoche.arrancar()
print(miCoche.estado())

# Acceso a un método de la clase Coche.
Nomenclatura del punto.
print("El coche está arrancado:" ,
miCoche.arrancar())

# Modificamos el valor de una propiedad
miCoche2.ruedas = 10
print("El coche2 tiene:" ,
miCoche2.ruedas, "ruedas.")
```

Veremos em tópicos posteriores, quando olharmos o encapsulamento, que não é conveniente acessar as propriedades e métodos da classe desta forma, mas no momento estamos simplesmente interessados no conceito de classe, objeto, atributos, métodos e na nomenclatura do ponto para acessá-los.

Para terminar este tópico, vamos ver outro exemplo de criação de classe e instanciação de objetos desta classe.

Vamos criar uma classe **Usuario** que deve ter os seguintes atributos:

- nome : string
- idade : number
- login : string
- password : string
- email : string
- telefone : number

E os seguintes métodos:

- Resumen(): Ele extrairá um resumo dos dados do usuário.
- cambiaEdad(): Isso nos dará a possibilidade de pedir ao usuário para entrar numa nova era.
- muestraEdad(): Mostrará a idade do usuário.

Finalmente, criaremos uma instância da classe Usuário, que chamaremos administrador e chamaremos os métodos da classe Usuário para este administrador usando a nomenclatura do ponto.

```
# CREACIÓN DE LA CLASE
class Usuario():

    # Declaración de atributos
    nombre = "Angel"
    edad = 47
    login = "admin"
    password = "1234"
    email = "angel@loquesea.com"
    telefono = 666666666
```

```
    # Declaración de métodos
    def resumen(self):  # self hace
referencia a la instancia de clase.
        print(f'Los datos del usuario
son:\n'
              f'Nombre: {self.nombre}\n'
              f'Edad: {self.edad}\n'
              f'Login: {self.login}\n'
              f'Password:
{self.password}\n'
              f'Email: {self.email}\n'
              f'Teléfono:
{self.telefono}')

    def cambiaEdad(self):
        edadIntroducida =
int(input("Introduce edad entre 18-
100:"))

        if 18 < edadIntroducida < 100:
            self.edad = edadIntroducida
            print("Edad introducida
correcta")
            return ""
        else:
            print("La edad introducida no
es correcta.")
            self.cambiaEdad()
            return ""


    def muestraEdad(self):
        print('La edad del usuario es:',
self.edad, 'años.')
        return ""

   # Creación de una instancia de la clase
Usuario a la que llamaremos administrador

administrador = Usuario()

    # Una vez creado el objeto administrador,
hacemos uso del método “resumen()” perteneciente
a la clase Usuario
```

```
administrador.resumen()
    # Usamos los métodos cambiaEdad() y
muestraEdad() de la clase Usuario.
print(administrador.cambiaEdad())
print(administrador.muestraEdad())
```

Se executarmos o programa, primeiro ele não recuperará os dados do usuário, depois nos pedirá para entrar na nova era e, finalmente, nos mostrará a idade entrada.

```
✓ class Usuario(): ···
···  Los datos del usuario son:
     Nombre: Angel
     Edad: 47
     Login: admin
     Password: 1234
     Email: angel@loquesea.com
     Teléfono: 666666666
     Edad introducida correcta

     La edad del usuario es: 55 años.
```