Computação II - Python
Aula 2 - Classe e Objetos

Carla A. D. M. Delgado

João C. P. da Silva

Dept. Ciência da Computação - UFRJ

Classes e Objetos

Classes - um tipo de dado genérico

Um pouco de filosofia...

- Para Aristóteles, temos a *forma* e a *matéria*. A forma seria uma ideia ou conceito que se materializa no mundo concreto. Por exemplo, temos a forma (ou conceito) **mesa** que podemos usar como uma referência para todas as mesas que econtrarmos em nossa vida.
- Uma mesa específica seria a materialização da forma de mesa.

Classes e Objetos

Classes - um tipo de dado genérico

Em programação, tanto em Python quanto em outras linguagens, temos os tipos de dados e os dados em si. É fácil fazer um paralelo entre o conceito de *forma* e os **tipos de dados**, sendo que cada dado concreto *materializa*, ou no termo adequado, **instancia** um tipo de dado específico.

- Instancias do tipo de dado **inteiro**: 2 , 10000
- Instancias do tipo de dados **string**: "Carla", "João"

Classes e Objetos

Classes - um tipo de dado genérico

Cada tipo de dado tem propriedades e comportamento pré-definidos. Por exemplo:

- Os números inteiros não tem parte decimal, podem ser positivos ou negativos e são operados com os operadores aritméticos.
- As strings tem a propriedade de serem um conjunto de caracteres alfanuméricos representados entre aspas, são imutáveis e tem um comportamento esperado quando sujeitas a qualquer operação que esteja definida pra elas.
- Listas tem a propriedade de serem mutáveis, e aumentam de tamanho quando sofrem uma operação de inserção ou concatenação.

Classes e Objetos

Classes - um tipo de dado genérico

- Classes são uma generalização do conceito de tipos dados. A classe pode ser vista como um recurso para que o programador defina seus próprios tipos de dados.
- Para isso, uma classe irá definir as propriedades (informações propriamente ditas) e o comportamento esperado que são comuns ao conjunto específico de dados que a classe representa.
- Chamamos de objeto cada instância de uma classe, ou seja, cada dado que corresponda ao tipo que a classe define.

Princípios da Programação Orientada a Objetos

Classes e objetos - exemplo

- Para fazer um programa de controle de contas de clientes em uma rede bancária, precisaremos armazenar informações sobre cada conta bancária.
- Criaremos uma classe para representar exatamente o tipo de dado **conta**, que agregará as informações e comportamentos básicos de uma conta corrente.
- A vantagem é que as informações da conta bem como os coportamentos esperados para ela estarão agrupados e serão programados de forma combinada.

Classes e objetos - exemplo - Conta bancária

- Em nosso primeiro exemplo, teremos como informação de uma conta apenas *saldo* e *numero da conta*

```
1 class Conta:
2     def __init__(self, numero, saldo=0):
3         self.saldo =saldo
4         self.numero = numero
```

Princípios da Programação Orientada a Objetos

Classes e objetos - exemplo - Conta bancária

```
1 class Conta:
2     def __init__(self, numero, saldo=0):
3         self.saldo =saldo
4         self.numero = numero
```

- Usamos a palavra **class** para indicar a definição de uma classe.
- A classe **Conta** tem dois **atributos** (*saldo* e *numero*)
- __init__ é um método especial, denominado **construtor** por ser chamado sempre que um objeto da classe é criado (instanciado).
- o método __init__ recebe um parâmetro chamado *self*, que indica o objeto que está sendo criado.

Classes e objetos - exemplo - Conta bancária

```
1 class Conta:
2     def __init__(self, numero, saldo=0):
3         self.saldo =saldo
4         self.numero = numero
```

- *self.saldo* indica o atributo *saldo* da conta que está sendo criada. Se não colocássemos *self.* na frente, saldo seria uma variável local e teria seu valor jogado fora quando o método acabasse de ser executado.

- no interpretador, podemos criar um objeto **conta** e atribuí-lo a uma variável. Para isso, usamos o nome da classe e os parâmetros indicados no método construtor __init__.

```
1 In [1]:  c1=Conta(1234, 100)
```

Classes e objetos - exemplo - Conta bancária

- Adicionaremos a nosso exemplo o comportamento da conta ao reagir às operações de *saque* e *deposito*, além da operação *resumo* para ver o saldo.

```
1  class Conta:
2      def __init__(self, numero, saldo=0):
3          self.saldo =saldo
4          self.numero = numero
5
6      def resumo(self):
7          print("CC numero: %s Saldo: %10.2f" % (self.numero, self.saldo))
8
9      def saque(self, valor):
10         if self.saldo>=valor:
11             self.saldo -= valor
12
13     def deposito(self, valor):
14         self.saldo += valor
```

- A classe **Conta** tem agora dois **atributos** (*saldo* e *numero*) e quatro **métodos** (__init__, *resumo*, *saque* e *deposito*)

Princípios da Programação Orientada a Objetos

Classes e objetos - exemplo - Conta bancária

- o primeiro parâmetro de todos os métodos de uma classe em Python tem que ser o *self*. Ele representa a instância sobre a qual o método atuará.
- os atribudos de um objeto tem seu valor preservado durante todo o tempo de vida do objeto. O tempo de vida de um objeto é o tempo em que alguma variável do seu programa o referencia.

```
1  class Conta:
2      def __init__(self, numero, saldo=0):
3          self.saldo =saldo
4          self.numero = numero
5
6      def resumo(self):
7          print("CC numero: %s Saldo: %10.2f" % (self.numero, self.saldo))
8
9      def saque(self, valor):
10         if self.saldo>=valor:
11             self.saldo -= valor
12
13     def deposito(self, valor):
14         self.saldo += valor
```

Classes e objetos - exemplo - Conta bancária

- Apesar de imprescindível na **definição** de cada método, não é necessário passar o *self* como parâmetro na hora de **chamar** um método, isso é feito automaticamente no interpretador Python.

```
In [1]:  c1=Conta(1234, 100)

In [2]: c1.resumo
Out[2]: <bound method Conta.resumo of <__main__.Conta object at 0x00000294A612DA58
     >>

In [3]: c1.resumo()
Out[3]: CC numero: 1234 Saldo:      100.00

In [4]: c1.saque(50)

In [5]: c1.resumo()
Out[5]: CC numero: 1234 Saldo:       50.00

In [6]: c1.deposito(200)

In [7]: c1.resumo()
Out[7]:CC numero: 1234 Saldo:      250.00
```

Classes e objetos - aplicação

- Usamos classes e objetos para facilitar a construção de programas mais complexos.
- Repare que nos métodos de *saque* e *deposito* não foi necessário passar o saldo como parâmetro, apenas o valor a ser sacado ou depositado.
- Esse efeito de memória ou permanência dos atributos é proposital para evitar passagens de muitos parâmetros ao lidarmos com informações complexas em nossos programas.
- A ideia é imitar o comportamento de objetos do mundo real: ao fazer um depósito ou saque em uma conta, você não precisa informar ao banco qual o saldo dessa conta.

Classes e objetos - Estudo Dirigido

Vamos abordar a situação de um banco que precise de um programa para controlar o saldo de seus correntistas.

- Cada conta corrente pode ter um ou mais clientes como titular.
- Para simplificar, diremos que o banco se interessa apenas pelo nome e CPF de seus correntistas.
- Toda conta corrente tem um saldo e uma lista de operações de saques e depósitos. Por enquanto, o banco não permite que os clientes saquem mais do que tem disponível na conta.

Classes e objetos - Estudo Dirigido

- Primeiro, vamos criar a classe **Cliente** para representar os dados e comportamento dos clientes que interessam ao banco.
- Crie a classe **cliente** com os atributos *CPF* e *nome* sendo iniciados no método construtor *init*.
- Salve a classe cliente em um arquivo chamado *clientes.py*
- Execute o arquivo. No interpretador, experimente criar um objeto da classe cliente.
- No interpretador, tente ver o nome e o CPF do objeto que foi criado.

Classes e objetos - Estudo Dirigido

- Como o programa para o banco vai ficar um pouco grande, sugerimos guardar cada classe em um arquivo separado. Essa separação não é obrigatória no Python, mas é uma boa maneira de organizar seu código.
- Vamos criar um arquivo para colocar nossos testes com as classes que criarmos para o banco. Chamaremos esse arquivo de *teste.py*. Salve este arquivo no mesmo diretório que você salvou o arquivo *clientes.py*.
- No arquivo *teste.py*, importe a classe **Cliente** que está no arquivo *clientes.py*.
- Ainda no arquivo *teste.py*, crie dois objetos **Cliente**, escolhendo nome e CPF para cada um deles. Você precisará também atribuir cada um destes dois objetos a uma variável distinta.

Princípios da Programação Orientada a Objetos

Classes e objetos - Estudo Dirigido

- Vamos agora criar a classe **Conta**, baseada no exemplo que vimos no início da aula. Nossa classe conta terá as mesmas informações e comportamento da classe conta que vimos no exemplo, porém terá também um ou mais correntistas, que são clientes do banco.
- Para que possamos ter um ou mais clientes como correntistas de uma conta, vamos usar uma lista de objetos da classe **Cliente**.
- Defina uma nova classe **Conta**, a partir do exemplo visto, que incorpore a lista de clientes como um atributo.
- Salve sua classe no arquivo *Contas.py*
- Altere *teste.py* para importar a classe **Conta**, e crie uma conta corrente para cada um dos clientes.
- Faça alguns testes no interpretador para observar a ação dos métodos sobre os atribudos dos objetos.

Classes e objetos - Estudo Dirigido

- O banco gostaria de oferecer a seus correntistas a possibilidade de emitir um extrato da conta, ou seja, uma lista com as operações realizadas. Altere a classe **Conta** de forma a incluir um atributo para armazenar essa lista, considerando o saldo inicial como um primeiro depósito na conta.
- Modifique os métodos depósito e saque para que essas operações fiquem registradas na lista.
- Crie um método na classe **Conta** para imprimir o extrato.
- Modifique o programa de teste para imprimir o extrato de cada conta.
- Faça testes no console.

Classes e objetos - Estudo Dirigido

- Faça com que a mensagem *saldo insuficiente* seja exibida caso haja tentativa de sacar uma quantia maior do que o saldo disponível.
- Modifique o método resumo da classe **Conta** para exibir também o nome e o CPF de cada correntista.
- Crie os clientes João e José no programa de testes.
- Crie uma nova conta, agora tendo João e José como clientes, e saldo igual a 500 no programa de testes.
- Faça testes no console.

Classes e objetos - Estudo Dirigido

- Para que seja possível gerenciar várias contas e vários clientes, precisamos de estruturas que nos permitam armazenar e manipular vários objetos de contas e vários objetos de clientes. Vamos criar uma classe **Banco** que tenha como atributos:
  - uma lista de clientes
  - uma lista de contas
  - o nome do banco
- Como métodos, nossa classe **Banco** terá uma operação para cadastrar um cliente, uma operação de abertura de conta (ou seja, para criar uma nova conta nesse banco), e outra para listar todas as contas.

Classes e objetos - Exercício

- Crie classes para representar estados e cidades. Cada estado tem um nome, uma sigla, e várias cidades. Cada cidade tem um nome e o número de habitantes.
- Crie um método que calcule mostre a população de um estado como a soma da população de suas cidades.
- Teste no console criando dois estados com algumas cidades cada um.

Classes e objetos - aplicação

- Classes representam abstrações de partes do problema que estamos tentando resolver com nosso programa. Muitos detalhes do contexto do problema são deixados de fora de nossos programas porque não contribuirão para sua resolução.
- A escolha dos atributos e métodos que irão compor uma classe, bem como a escolha de que classes teremos em nosso programa depende de como o programador entende o problema e como deseja resolvê-lo, ou seja, pode variar muito de pessoa para pessoa. A isso chamamos *modelagem do problema*.
- Fazer bons modelos é algo que se aprende com a experiência, portanto é importante praticar. Errar fará parte do processo.

Classes e objetos

Você não sabia, mas ao usar listas e strings, estava trabalhando com classes e objetos, pois estes tipos de dados são definidos como classes.

- as funções que vimos para as listas e strings na verdade são os métodos definidos para a classe lista (reverse, append, etc) e para a classe string, respectivamente.
- ao criar uma lista ou um string, estamos instanciando ou criando uma instancia dessas classes, ou seja, objetos.

Princípios da Programação Orientada a Objetos

A Programação Orientada a Objetos (POO) é um paradigma de programação que tem muitas coisas em comum com a programação modular...

- é baseada em comandos e tipos de dados
- organiza o programa em blocos de código
- usa fortemente todos os conceitos que já vimos anteriormente

Princípios da Programação Orientada a Objetos

A Programação Orientada a Objetos (POO) é um paradigma de programação que tem muitas coisas em comum com a programação modular,

porém

também tem diferenças significativas:

- o código é organizado em definições de hierarquias de **classes** (e não em funções convencionais).
- assim como uma função pode ser usada várias vezes, uma classe pode ser instanciada várias vezes. Cada instância é chamada de **Objeto** (daí o nome POO).
- um programa em execução é como uma teia de objetos que se comunicam entre si através de mensagens (e não como uma função principal que chama de forma articulada outras funções).

Princípios da Programação Orientada a Objetos

- Por ser uma técnica completa de programação, o paradigma da Orientação a Objetos é um assunto extenso e complexo.
- Seu domínio está além dos objetivos deste curso. Nossa intenção aqui é uma compreensão básica de seus princípios e a utilização de seus principais conceitos em nosso dia a dia como programadores.

Referências

Para saber mais

1. Livro: **Introdução à Programação com Python - Algoritmos e lógica de programação para iniciantes**. Autor: Nilo Ney Coutinho Menezes

Autores

- **Carla Delgado** ▸ Lattes
- **João C. P. da Silva** ▸ Lattes

Computação II - Python
Aula 2 - Classe e Objetos

Carla A. D. M. Delgado

João C. P. da Silva

Dept. Ciência da Computação - UFRJ