MC-102 — Aula 17
Classes e objetos

Instituto de Computação – Unicamp

1 de Outubro de 2019

Classes

- Uma classe é um mecanismo da linguagem Python (e linguagens orientada a objetos em geral) para agrupar várias variáveis, que inclusive podem ser de tipos diferentes, mas que dentro de um contexto, fazem sentido estarem juntas.
- Uma classe também pode agrupar métodos (funções) em conjunto com as variáveis.
- Os métodos da classe servem em geral para manipular os dados (variáveis) da classe.
- Exemplos de uso de classes:
  - Criar um registro de alunos para guardar os dados: (nome, RA, médias de provas, médias de labs, etc...)
  - Criar um registro de pacientes para guardar os dados: (Nome, endereço, histórico de doenças, etc...)

Declarando uma Classe

- Para criarmos uma classe usamos a palavra chave **class** da seguinte forma:

```
class nome_da_classe:
  def __init__(self):
      self.variáveis_da_instância
  método_1
  ...
  método_m
```

- Cada variável e método da classe deve ser declarado com identação dentro da classe.

Exemplo:

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]
```

Declarando uma Classe

- A criação de uma classe é entendida como a criação e um novo tipo (como int, float, list,..).
- É possível criar então instâncias desta classe. Uma instância de uma classe é conhecido como **objeto** daquela classe.
- Pode-se criar um objeto e atribuí-lo para uma variável com a seguinte sintaxe:

```
nome_var = nome_da_classe()
```

Exemplo:

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]
a = Aluno() #Cria um novo objeto do tipo Aluno com dois campos, nome e notas
b = Aluno() #Cria um novo objeto do tipo Aluno com dois campos, nome e notas
```

Utilizando os campos de uma classe

- Podemos acessar individualmente os campos de um determinado objeto como se fossem variáveis normais. A sintaxe é:

```
nome_da_variável.nome_do_campo
```

- Os campos individuais de um objeto tem o mesmo comportamento de qualquer variável.
  - Isto significa que todas operações válidas para variáveis de um tipo são válidas para um campo do mesmo tipo.

Utilizando os campos de uma classe

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]

a = Aluno()
b = Aluno()
a.nome="Joao"
a.notas.append(7.6)
b.nome="Maria"
b.notas.append(8.5)
print(a.nome, a.notas)
print(b.nome, b.notas)
```

A saída será:

```
Joao [7.6]
Maria [8.5]
```

Lendo e Escrevendo Classes

- A leitura dos campos de um objeto a partir do teclado deve ser feita campo a campo, como se fossem variáveis independentes.
- A mesma coisa vale para a escrita, que deve ser feita campo a campo.

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]

a = Aluno()
b = Aluno()

a.nome=input("Digite um nome:")
a.notas.append(float(input("Digite uma nota:")))

b.nome=input("Digite um nome:")
b.notas.append(float(input("Digite uma nota:")))

print(a.nome, a.notas)
print(b.nome, b.notas)
```

Lista de Objetos

Uma lista pode conter quaisquer objetos, inclusive objetos de classes criadas pelo programador:

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]
a = Aluno()
b = Aluno()
a.nome="AA"
b.nome="BB"
lista = [a, b]
for i in lista:
    print(i.nome)
```

A saída será:

```
AA
BB
```

Funções e Objetos

- Objetos podem ser usados tanto como parâmetros em funções bem como em retorno de funções.
- Neste caso o comportamento de objetos de uma classe criada é o mesmo de outros tipos vistos.

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]

def imprimeAluno(a):
    print("Nome: ", a.nome)
    print("Notas: ", a.notas)

def leAluno():
    a = Aluno()
    a.nome=input("Digite Nome: ")
    a.notas.append(float(input("Digite Nota: ")))
    return a
a = leAluno()
b = leAluno()
imprimeAluno(a)
imprimeAluno(b)
```

Métodos da Classe

- É possível declarar funções dentro de uma classe (chamamos tais funções de métodos).
- Para executar um método basta usar a mesma notação para acessar um campo do objeto.
- É comum um método de uma classe acessar os campos da classe, por isso o primeiro parâmetro do método deve ser **self**, que se referencia para o objeto corrente.

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]

    def printAluno(self):
        print("Nome: ", self.nome)
        print("Notas: ", self.notas)

    def input(self):
        self.nome=input("Digite Nome: ")
        self.notas.append(float(input("Digite Nota: ")))
```

Métodos da Classe

- No exemplo abaixo, tanto **a** quanto **b** possuem os métodos **printAluno** e **input**.
- Ao executarmos **a.input()** automaticamente é passado **a** como parâmetro para **self**, e por isso os dados são lidos corretamente tanto para **a** quanto para **b**.

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]

    def printAluno(self):
        print("Nome: ", self.nome)
        print("Notas: ", self.notas)

    def input(self):
        self.nome=input("Digite Nome: ")
        self.notas.append(float(input("Digite Nota: ")))

a = Aluno()
b = Aluno()
a.input()
b.input()
```

Métodos da Classe

Exemplo completo:

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]

    def printAluno(self):
        print("Nome: ", self.nome)
        print("Notas: ", self.notas)

    def input(self):
        self.nome=input("Digite Nome: ")
        self.notas.append(float(input("Digite Nota: ")))

a = Aluno()
b = Aluno()
a.input()
b.input()
a.printAluno()
b.printAluno()
```

Metodos e classes até agora

```
a = [1,2,3]  # cria um objeto da classe lista
a.append(44)  # metodo append
a.sort()          # metodo

ff = open('arquivo.txt', 'r')  # cria um objeto da classe file
ff.write(saida)     # metodo
ff.close()        # outro metodo

d = {'A': 3, 'B': 4}  # cria um objeto da classe dicionario
d.items()     # metodo
```

Exemplo

Vamos criar um programa que simula um cadastro de alunos de uma turma.

- Para representar os alunos criamos a classe abaixo que além dos campos **nome** e **notas** possui métodos para:
  - (a) imprimir os dados do aluno.
  - (b) ler o nome de um aluno.
  - (c) incluir uma nota.

```
class Aluno:
    def __init__(self):
        self.nome=""
        self.notas=[]

    def printAluno(self):
        print("Nome: ", self.nome)
        print("Notas: ", self.notas)

    def inputAluno(self):
        self.nome=input("Digite Nome: ")

    def incluiNota(self):
        self.notas.append(float(input("Digite a Nota:")))
```

Exemplo

- Para representar o cadastro criamos a classe abaixo que contém um campo **cadastro** que é uma lista de **Alunos**.
- Ela possui métodos para:
  - (a) inclusão de um aluno no cadastro.
  - (b) exclusão de um aluno do cadastro.
  - (c) impressão dos alunos no cadastro.

```
class Cadastro:
    def __init__(self):
        self.cadastro=[]

    def incluiAluno(self,a):
        self.cadastro.append(a)

    def excluiAluno(self, a):
        for i in self.cadastro:
            if i.nome == a.nome:
                self.cadastro.remove(i)

    def printCad(self):
        print("Imprimindo Cadastro")
        for a in self.cadastro:
            a.printAluno()
```

Exemplo

- Com estas classes criadas podemos por exemplo criar o programa abaixo:

```
cad = Cadastro()
a = Aluno()
b = Aluno()
a.nome="AA"
b.nome="BB"
cad.incluiAluno(a)
cad.incluiAluno(b)
cad.printCad()
c = Aluno()
c.nome = "AA"
cad.excluiAluno(c)
cad.printCad()
```

Saída :

```
Imprimindo Cadastro
Nome: AA
Notas: []
Nome: BB
Notas: []
Imprimindo Cadastro
Nome: BB
Notas: []
```

Exercício

- Crie um novo tipo de classe para representar coordenadas no plano cartesiano.
- Crie uma função para imprimir um ponto do tipo criado.
- Crie uma função para cada uma destas operações: soma de dois pontos, subtração de dois pontos, multiplicação por um escalar.