



- Classes na Cozinha
- Primeira Classe
- Classes c/ Método e Var
- Módulos e Import
- Classes e Listas
- Buscar objetos em lista
- Usando índices e listas

Python

Lógica de Programação

Classes, Módulos e Imports com Python

01 Classes na “Cozinha”

- Linguagens como Java são baseadas em Programação Orientada à Objetos (POO)
- A melhor forma que eu consigo explicar o conceito de **Classes** é usando conceitos da culinária. **Classe** é como se fosse uma forma de um bolo, como o da imagem a seguir.
- Uma **Classe** em Python agrupa métodos (funções) e variáveis (atributos). No exemplo a seguir, entre as linhas 1 e 5, temos uma classe chamada Conta, com 4 variáveis. Note que, cada variável eu imagino que é uma parte da forma do bolo (Classe).



```
1 class Conta:
2     agencia = 0
3     conta = 0
4     saldo = 0.0
5     titular = ''
6
7 contaDaniel = Conta()
8 contaNathaly = Conta()
```



- Como toda forma de bolo, a partir dela geramos bolos. Traduzindo para programação:
 - Classe = forma do bolo
 - Instância ou Objetos = bolo gerado a partir da forma
- Na linha 6 acima, temos um objeto (bolo) cuja variável se chama contaDaniel. Na linha 7, agora você já sabe o que é.

02 Criando sua Primeira Classe

- Abaixo (linhas 1 e 2) temos a criação de uma classe vazia (ou seja, sem nada) chamada Conta, enquanto que, na linha 4, criamos um objeto chamado novaConta.



Note que, classe inicia sempre com Maiúsculo, enquanto métodos com minúsculos. E, quando a classe ou método estiver vazio, precisamos por a palavra pass. Detalhes em <https://legacy.python.org/dev/peps/pep-0008/>.

```
main.py x
1 class Conta:
2     pass
3
4 novaConta = Conta()
```

03 Classes com Métodos e Variáveis

- Na classe a seguir, temos uma classe Conta com uma variável chamada cotacaoDolar e um método chamado converterDolares, que por sua vez, recebe o valor em reais e dá um print no valor em dólares.
 - O que fez esse self no método? Em métodos dentro de classes, obrigatório!
 - Na linha 9, criamos um objeto chamado novaConta
 - Na linha 10, chamamos o método que converte usando o objeto da linha 9

```
1 class Conta:
2
3     cotacaoDolar = 4.9
4
5     def converterDolares(self, valorReais):
6         print(valorReais * self.cotacaoDolar)
7         print(valorReais * 4.9)
8
9     novaConta = Conta()
10    novaConta.converterDolares(100)
```

- No código acima, nas linhas 6 e 7, imprimimos o valor em reais de duas maneiras (sem variável e com variável). Prefiro a opção da linha 6, pois facilita a manutenção
 - Se amanhã o valor do dólar for 5.0, ajusto apenas a variável da linha 3
 - Note que, na linha 6, usamos self.cotacaoDolar
 - Self é obrigatório, lembre-se!
- E se, ao invés de imprimir (print), eu queira retornar (melhor opção), como seria? Código a seguir.
 - Note que, na linha 9, imprimimos o retorno do método
 - Alternativamente, na linha 11, atribuímos o retorno a uma variável e na linha 12, imprimimos a variável

```

1 ▼ class Conta:
2
3     cotacaoDolar = 4.9
4
5 ▼ def converterDolares(self, valorReais):
6     return valorReais * self.cotacaoDolar
7
8 novaConta = Conta()
9 print(novaConta.converterDolares(100))
10
11 valorEmDolares = novaConta.converterDolares(100)
12 print(valorEmDolares)

```

- Vamos agora criar uma classe mais próxima a realidade? Uma classe geralmente tem, ao menos, número da agência, número da conta, titular e o saldo, concordam? Exemplo abaixo.

```

1 ▼ class Conta:
2     agencia = 0
3     conta = 0
4     saldo = 0.0
5     titular = ''
6
7 ▼ def __init__(self, a, c, s, t):
8     self.agencia = a
9     self.conta = c
10    self.saldo = s
11    self.titular = t

```

- Mas, o que “danado” é esse método init? Ele é um método que comumente chamamos de **inicializador** ou **construtor**. Ou seja, com ele, eu consigo já criar um projeto com os valores, como vou mostrar a seguir.
 - Na página anterior, sem construtor, lembram que criamos um objeto da seguinte maneira: `novaConta = Conta()`

! No construtor acima, a linha 7 criamos um construtor, que é um método com nome `__init__`. Como todo método de classe, o primeiro parâmetro precisa ser o self. Após o self, temos 4 parâmetros (a,c,s,t) que guardam os valores informados pelo usuário. E, entre as linhas 8 e 11, atribuímos às variáveis das linhas 2 a 5 os valores recebidos pelos 4 parâmetros. Por exemplo, na linha 8, a variável de classe `agencia`, guarda o valor do parâmetro `a`.

- Enfim, agora, com um construtor, criamos o objeto da maneira das linhas 13 e 16 abaixo.
 - E, nas linhas 14 e 17, apresentamos respectivamente o saldo de Abella (100) e de Arthur (1000).

```

12
13 contaDaniel = Conta(1,123,100,'Abella')
14 print(contaDaniel.saldo)
15
16 contaArthur = Conta(1,124,1000,'Arthur')
17 print(contaArthur.saldo)

```

04 Módulos e Import

- Até o presente momento, tudo o que fazemos é dentro do arquivo `main.py`. Conforme os sistemas sejam maiores, isto não é escalável. Para isto, usamos o conceito de módulos.
 - Em resumo, módulos são arquivos com extensão `py` que podem armazenar métodos, classes e variáveis
 - No exemplo a seguir, criamos um módulo chamado `calculadora`, que basicamente é um arquivo `calculadora.py` e possui um método `soma`.
 - Não tem self aqui porque não está dentro de uma classe.

```

calculadora.py ×
1 ▼ def soma(x,y):
2     return x+y

```

- E, como faço para usar um módulo? No código a seguir, na linha 1, importamos o módulo `calculadora` por meio da palavra-chave import
 - Para usar, chamamos de uma das 3 seguintes maneira:
 - módulo.método
 - módulo.variável
 - módulo.Classe

```

main.py ×
1 import calculadora
2
3 print(calculadora.soma(2,2))

```

- Se o módulo `calculadora` acima tivessem mais outros métodos, chamaríamos da mesma maneira que chamamos a `soma`.

! No exemplo acima importamos o módulo inteiro (ou seja, tudo que tiver dentro do módulo). Mas, se eu quiser importar apenas um método de um módulo? **from modulo import método**

```

1 from calculadora import soma
2
3 print(soma(2,2))

```

Se você importa por método, podemos chamar soma diretamente. Ou seja, não precisa fazer `calculadora.soma(2,2)`, como tínhamos feito anteriormente.

- Uma excelente ideia é, colocar as classes dentro de módulos. Note que, no código a seguir, criamos um módulo chamado `conta`, que possui um construtor / inicializador e um método `sacar`. Que, recebe um montante e diminui do saldo.

conta.py ×

```
1 class Conta:
2     agencia = 0
3     conta = 0
4     saldo = 0.0
5     titular = ''
6     cotacaoDolar = 4.9
7
8 def __init__(self, a, c, s, t):
9     self.agencia = a
10    self.conta = c
11    self.saldo = s
12    self.titular = t
13
14 def sacar(self, montante):
15     self.saldo = self.saldo - montante
```

- No código abaixo, explico como usar o módulo criado anteriormente.
 - Linha 1:** Importamos o módulo inteiro
 - Linha 3:** Criamos um objeto para Daniel com saldo 1000
 - Linha 4:** Chamamos o método sacar na variável novaConta
 - Linha 5:** Apresentamos o novo saldo (900)

main.py ×

```
1 import conta
2
3 novaConta = conta.Conta(123,442233,1000,'Daniel')
4 novaConta.sacar(100)
5 print(novaConta.saldo)
```

05 Classes e Listas

- Agora que já sei como funcionam as classes, posso adicionar um objeto à uma lista? Por exemplo, termos uma lista de contas de uma dada agência bancária? Sim, no exemplo a seguir apresento este exemplo.

main.py ×

```
1 import conta
2
3 listaContasAgencia = [conta.Conta(123,442232,100,'Daniel')]
4
5 novaConta2 = conta.Conta(123,442233,101,'Nathaly')
6 novaConta3 = conta.Conta(123,442234,102,'Arthur')
7
8 listaContasAgencia.append(novaConta2)
9 listaContasAgencia.append(novaConta3)
10
11 print(len(listaContasAgencia))
12
13 for con in listaContasAgencia:
14     print('Agencia ',con.agencia, ' Conta ', con.conta)
```

- Linha 1:** Importamos o módulo inteiro
- Linha 3:** Criamos uma lista normalmente
- Linha 3:** Adicionamos “de cara” um objeto para Daniel
 - Não era necessário, mas quis mostrar como seria

- Linhas 5 e 6:** Criamos 2 objetos
 - Não foram inseridos à lista ainda
- Linhas 8 e 9:** Adicionamos os 2 objetos a lista com append
 - Não mudou nada 😊
- Linha 11:** Apresentamos o tamanho da lista (3)
- Linhas 13 e 14:** Apresentamos todos os objetos que estão na listas

06 Buscar objetos em lista

- Quando usamos listas com *int*, *float*, *bool* ou *str*, a busca era bem simples. Agora, quando a lista contém objetos, o negócio “é mais embaixo”. O exemplo completo está a seguir.
 - Linha 12:** Criamos uma variável vazia (None)
 - Linha 15:** Verifica se a variável titular do objeto (con) é igual a ‘Daniel’
 - Linha 16:** Atribuímos o objeto (con) à variável da linha 12
 - Linha 17:** Se eu achar o objeto que eu preciso, por que eu preciso ir até o final da lista? “Meto” um break para parar a busca.
 - Linhas 19 e 20:** Apresenta os dados da conta buscada (de Daniel)

main.py ×

```
1 import conta
2
3 listaContasAgencia = []
4
5 novaConta1 = conta.Conta(123,442232,100,'Daniel')
6 novaConta2 = conta.Conta(123,442234,102,'Arthur')
7
8 listaContasAgencia.append(novaConta1)
9 listaContasAgencia.append(novaConta2)
10 print(len(listaContasAgencia))
11
12 contaDaniel = None
13
14 for con in listaContasAgencia:
15     if con.titular == 'Daniel':
16         contaDaniel = con
17         break
18
19 print('agencia ',contaDaniel.agencia)
20 print('saldo ',contaDaniel.saldo)
```

07 Usando Lista de Objetos com Índices

```
1 import conta
2
3 listaContasAgencia = []
4 novaConta1 = conta.Conta(123,442232,100,'Daniel')
5 listaContasAgencia.append(novaConta1)
6 contaDaniel = None
7
8 for i in range(len(listaContasAgencia)):
9     if listaContasAgencia[i].titular == 'Daniel':
10         contaDaniel = listaContasAgencia[i]
11         break
12
13 print('saldo ',contaDaniel.saldo)
```