

AC-309 – Atividades Complementares

(Engenharias Biomédica e de Telecomunicações)



5. POO – Programação Orientada a Objetos em Python

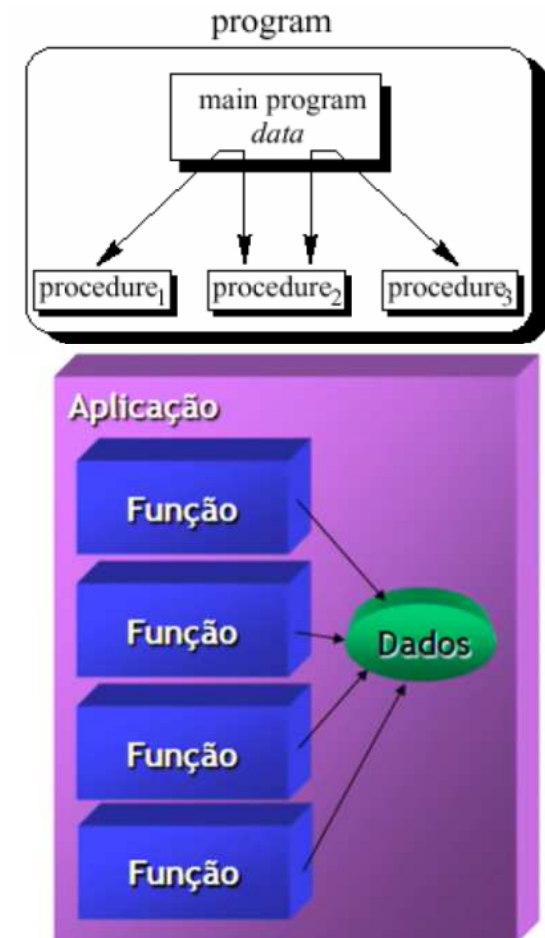
Prof. Evandro Luís Brandão Gomes

Introdução à Orientação a Objetos em Python

Comparativo: Programação Estruturada (procedural) x Orientada a Objetos

Estruturada ou Procedural:

- A escrita do código é focada na lógica para a resolução do problema;
- Em qualquer parte do código é possível utilizar o valor de uma variável (sem necessidade de permissão);
- Os procedimentos ou funções são combinados para prover a funcionalidade desejada;
- O programa pode ser visto como uma sequência de chamadas de procedimento ou funções;



Introdução à Orientação a Objetos em Python

Comparativo: Programação Estruturada (procedural) x Orientada a Objetos

Orientada a Objetos:

- Permite dividir o código em camadas (cada uma com suas responsabilidades);
- Adiciona segurança a aplicação através do **encapsulamento** dos dados (classes) e definir o que é privado (somente a classe acessa) e o que é público (todos acessam)



Introdução à Orientação a Objetos em Python

Vantagens da POO em relação à programação estruturada

- Maior índice de reaproveitamento de código
- Maior facilidade de manutenção
- Menos código gerado
- Maior confiabilidade no código
- Maior facilidade de gerenciamento do código (reduz grandes problemas para problemas menores);

Introdução à Orientação a Objetos em Python

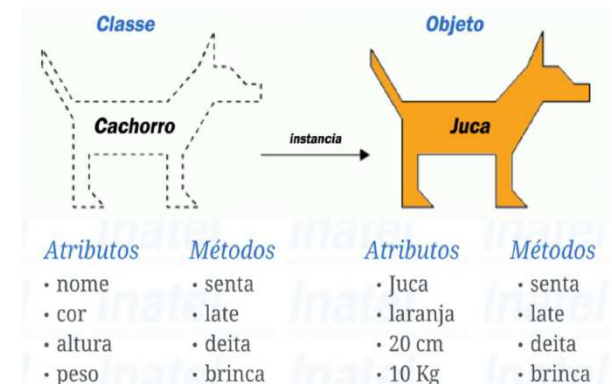
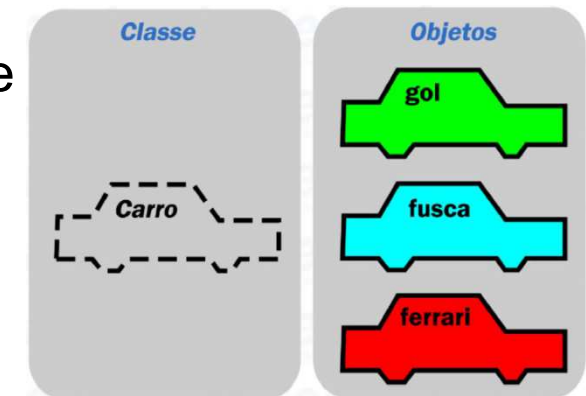
Classes e Objetos

- Uma classe **descreve um conjunto de objetos semelhantes**
- Atributos e métodos que resumem as características comuns de vários objetos
- Objeto constitui uma entidade concreta com tempo e espaço de existência
- Classe é tão-somente uma abstração
- Em termos de programação, definir uma classe significa formalizar um tipo de dado e todas as operações associadas a esse tipo, enquanto declarar objetos significa criar variáveis do tipo definido

Introdução à Orientação a Objetos em Python

Classes e Objetos

- Classe é um template (“forma”) para a criação de objetos
- Uma classe especifica os tipos de dados (atributos) e operações (métodos) suportadas por um conjunto de objetos
- Um objeto é uma *instância* de uma classe
- Criação de um objeto a partir de uma classe é chamada de **instanciação**
- É muito comum que em um programa existam várias instâncias de uma mesma classe



Introdução à Orientação a Objetos em Python

Tipos de Acesso (visibilidade)

- Uma classe pode definir o tipo de acesso à seus membros (atributos e métodos)

- **Público**

- Atributo ou método da classe pode ser acessado por todas as demais entidades do sistema

- **Privado**

- Atributo ou método da classe pode ser acessado somente por métodos da própria classe

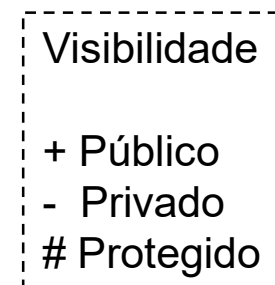
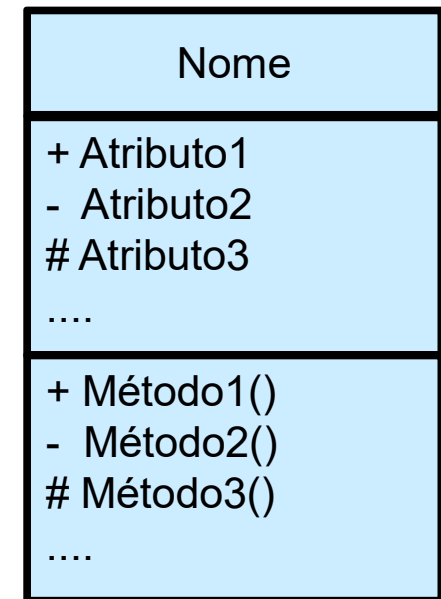
- **Protegido**

- Atributo ou método da classe pode ser acessado somente por classes da mesma hierarquia

Introdução à Orientação a Objetos em Python

Representação Gráfica de uma Classe

- O Diagrama de Classe, é um dos vários tipos de diagramas oferecidos pela UML (*Unified Modeling Language – Linguagem de Modelagem Unificada*). *Este diagrama tem como função descrever os vários tipos de classes/objetos de um sistema e o relacionamento entre eles.*
- Permite visualizar uma abstração independente de qualquer linguagem de implementação específica, dando ênfase às partes mais importantes: nome da classe, atributos e métodos (operações)



Introdução à Orientação a Objetos em Python

Exemplo:

Um programa para movimentação de conta bancária. É bem fácil de perceber que uma entidade extremamente importante para este programa é a CONTA do usuário.

O que toda CONTA tem?

- Número da conta;
- Nome do dono da conta;
- Saldo
- Limite

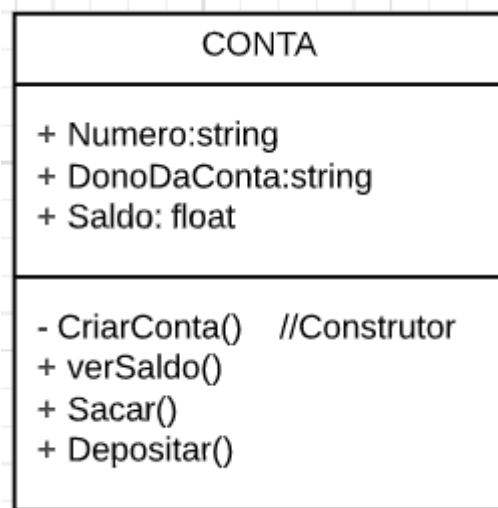
O que toda CONTA faz?

- Sacar
- Depositar
- Verificar Saldo
- Transferir valor

Introdução à Orientação a Objetos em Python

Exemplo: Implementando uma classe CONTA em Python:

Diagrama de Classe:



Obs: em python:

Construtor da classe → método “`__init__()`”

Visibilidade:

+ → default

- → colocar antes do método ou atributo “`__`”

Referência ao objeto instanciado → “`self`”

Diagrama feito em url: <http://lucidchart.com>

Introdução à Orientação a Objetos em Python

Exemplo: Implementando uma classe CONTA em Python:

```
1 #####
2 # Exemplo de uso de POO (classes e objetos) em python
3 # com métodos e atributos públicos
4 #####
5 class CONTA:
6     def __init__(self, num, nome, valor): #metodo construtor
7         self.Numero = num
8         self.DonoDaConta = nome
9         self.Saldo = valor
10        print("Conta do %s criada com saldo inicial de %.2f" %(self.DonoDaConta,self.Saldo))
11    def verSaldo(self):
12        print("Saldo = %.2f" %self.Saldo)
13    def Sacar(self,valor):
14        self.Saldo -= valor
15        print("Saque de %.2f realizado" %valor)
16    def Depositar(self, valor):
17        self.Saldo += valor
18        print("Deposito de %.2f realizado" %valor)
```

CONTA
+ Numero:string + DonoDaConta:string + Saldo: float
- CriarConta() //Construtor + verSaldo() + Sacar() + Depositar()

Obs: em python:

Construtor da classe → método “__init__()”

Visibilidade:

+ → default

- → colocar antes do método/atributo “_”

→ colocar antes do método/atributo “_”

Referência ao objeto instanciado → “self”

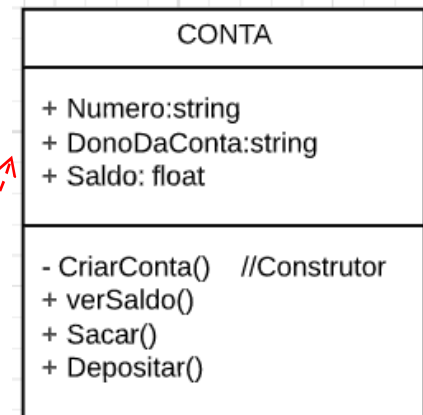
Introdução à Orientação a Objetos em Python

Exemplo: Implementando uma classe CONTA em Python:

```

19
20
21 conta1 = CONTA(100,"joão da Silva",1500.00)
22
23 conta1.Depositar(1000.00)
24 conta1.verSaldo()
25 conta1.Sacar(500.00)
26 conta1.verSaldo()
27 conta1.Saldo = 5000.00
28 conta1.verSaldo()

```



```

Conta do joão da Silva criada com saldo inicial de 1500.00
Deposito de 1000.00 realizado
Saldo = 2500.00
Saque de 500.00 realizado
Saldo = 2000.00
Saldo = 5000.00

```

Process finished with exit code 0

Essa operação é permitida porque o atributo "Saldo" foi declarado como público.

Introdução à Orientação a Objetos em Python

Exemplo: Mesmo exemplo, porém com os atributos privados:

```

1 #####
2 # Exemplo de uso de POO (classes e objetos) em python
3 # com metodos publicos e atributos privados
4 #####
5 class CONTA:
6     def __init__(self, num, nome, valor): #metodo construtor
7         self.__Numero = num
8         self.__DonoDaConta = nome
9         self.__Saldo = valor
10        print("Conta do %s criada com saldo inicial de %0.2f" %(self.__DonoDaConta, self.__Saldo))
11    def verSaldo(self):
12        print("Saldo = %0.2f" %self.__Saldo)
13    def Sacar(self, valor):
14        self.__Saldo -= valor
15        print("Saque de %0.2f realizado" %valor)
16    def Depositar(self, valor):
17        self.__Saldo += valor
18        print("Deposito de %0.2f realizado" %valor)

```

CONTA
- Numero:string - DonoDaConta:string - Saldo: float
- CriarConta() //Construtor + verSaldo() + Sacar() + Depositar()

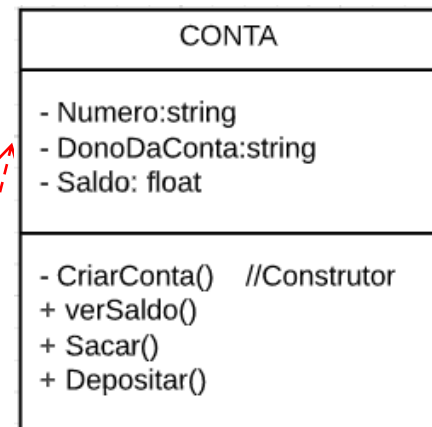
Introdução à Orientação a Objetos em Python

Exemplo: Implementando uma classe CONTA em Python:

```

19
20
21 conta1 = CONTA(100,"joão da Silva",1500.00)
22
23 conta1.Depositar(1000.00)
24 conta1.verSaldo()
25 conta1.Sacar(500.00)
26 conta1.verSaldo()
27 conta1.Saldo = 5000.00
28 conta1.verSaldo()

```



Conta do João da Silva criada com saldo inicial de 1500.00
 Depósito de 1000.00 realizado
 Saldo = 2500.00
 Saque de 500.00 realizado
 Saldo = 2000.00
 Saldo = 2000.00

O atributo `__Saldo` é privado, portanto não tem efeito se acessado fora da classe.

OBS: essa é a forma mais segura de implementar os atributos de uma classe, visto que somente o objeto referenciado tem acesso aos seus atributos (encapsulamento).

Introdução à Orientação a Objetos em Python

Outros exemplos: 1- Atributos de uma classe fora do método construtor.

```
1 #####
2 # Exemplo de uso de POO (classes e objetos) em python
3 #####
4 class CONTA:
5     contAcessos = 0
6     def __init__(self, num, nome, valor): #metodo construtor
7         self.__Numero = num
8         self.__DonoDaConta = nome
9         self.__Saldo = valor
10        print("Conta do %s criada com saldo inicial de %0.2f" %(self.__DonoDaConta,self.__Saldo))
11    def verSaldo(self):
12        self.contAcessos += 1
13        print("Saldo = %0.2f" %self.__Saldo)
14    def Sacar(self,valor):
15        self.__Saldo -= valor
16        self.contAcessos += 1
17        print("Saque de %0.2f realizado" %valor)
18    def Depositar(self, valor):
19        self.__Saldo += valor
20        self.contAcessos += 1
21        print("Deposito de %0.2f realizado" %valor)
```


Introdução à Orientação a Objetos em Python

Outros exemplos: 2 – Métodos podem retornar valor.

```
1 #####
2 # Exemplo de uso de POO (classes e objetos) em python
3 #####
4 class CONTA:
5     def __init__(self, num, nome, valor): #metodo construtor
6         self.__Numero = num
7         self.__DonoDaConta = nome
8         self.__Saldo = valor
9         print("Conta do %s criada com saldo inicial de %0.2f" %(self.__DonoDaConta,self.__Saldo))
10    def verSaldo(self):
11        return self.__Saldo
12    def Sacar(self,valor):
13        self.__Saldo -= valor
14        print("Saque de %0.2f realizado" %valor)
15    def Depositar(self, valor):
16        self.__Saldo += valor
17        print("Deposito de %0.2f realizado" %valor)
18
19    conta1 = CONTA(100,"joão da Silva",1500.00)
20    conta1.Depositar(1000.00)
21    print("Saldo = %0.2f" %conta1.verSaldo())
```

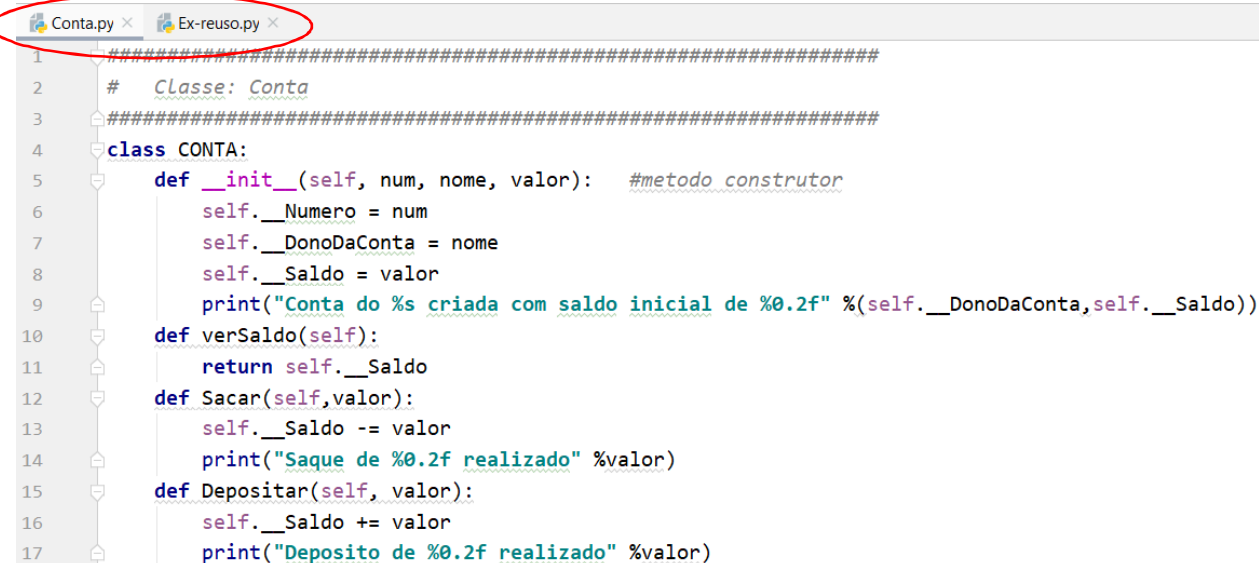

Introdução à Orientação a Objetos em Python

Outros exemplos: 3 – Instanciação de vários objetos da mesma classe.

```
1 #####
2 # Exemplo de uso de POO (classes e objetos) em python
3 #####
4 class CONTA:
5     def __init__(self, num, nome, valor): #metodo construtor
6         self.__Numero = num
7         self.__DonoDaConta = nome
8         self.__Saldo = valor
9         print("Conta do %s criada com saldo inicial de %0.2f" %(self.__DonoDaConta, self.__Saldo))
10    def verSaldo(self):
11        return self.__Saldo
12    def Sacar(self, valor):
13        self.__Saldo -= valor
14        print("Saque de %0.2f realizado" %valor)
15    def Depositar(self, valor):
16        self.__Saldo += valor
17        print("Deposito de %0.2f realizado" %valor)
18
19    conta1 = CONTA(100, "joão da Silva", 1500.00)
20    conta2 = CONTA(200, "Joaquim José da Silva Xavier", 500.00)
21    conta1.Depositar(1000.00)
22    conta2.Depositar(5000.00)
```

Introdução à Orientação a Objetos em Python

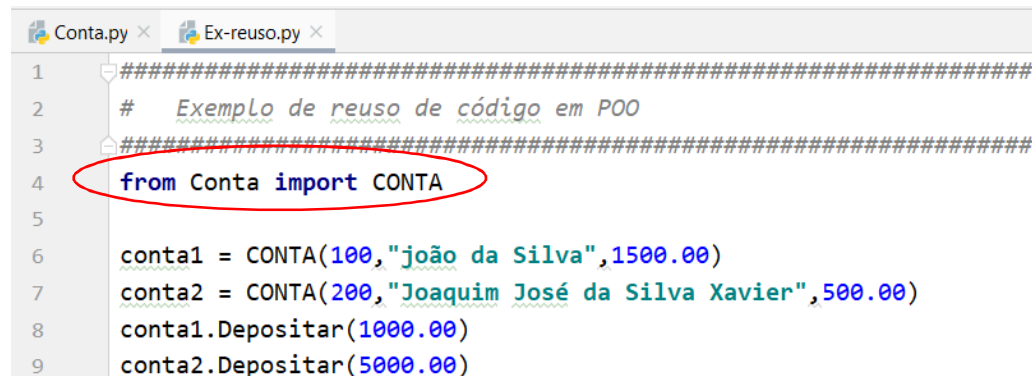
Reutilização de código:



```

1 #####
2 # Classe: Conta
3 #####
4 class CONTA:
5     def __init__(self, num, nome, valor):  #metodo construtor
6         self.__Numero = num
7         self.__DonoDaConta = nome
8         self.__Saldo = valor
9         print("Conta do %s criada com saldo inicial de %0.2f" %(self.__DonoDaConta, self.__Saldo))
10    def verSaldo(self):
11        return self.__Saldo
12    def Sacar(self, valor):
13        self.__Saldo -= valor
14        print("Saque de %0.2f realizado" %valor)
15    def Depositar(self, valor):
16        self.__Saldo += valor
17        print("Deposito de %0.2f realizado" %valor)

```



```

1 #####
2 # Exemplo de reuso de código em POO
3 #####
4 from Conta import CONTA
5
6 conta1 = CONTA(100, "João da Silva", 1500.00)
7 conta2 = CONTA(200, "Joaquim José da Silva Xavier", 500.00)
8 conta1.Depositar(1000.00)
9 conta2.Depositar(5000.00)

```

Introdução à Orientação a Objetos em Python

Exercícios:

1) Implemente a classe Funcionário. Um funcionário tem um nome e um Salário como atributo. Os métodos devem ser: cadastraFuncionário, aumentarSalario (que aumente o salário do funcionário em uma certa porcentagem digitada como entrada de dados) e mostrarSalario.

Exemplo de uso:

```
func1 = funcionario("João da Silva" , 10000,00)
func1.aumentarSalario(10)
func1.mostraSalario()
```

Faca um programa que teste o método da classe, criando pelos menos 3 funcionários.

Introdução à Orientação a Objetos em Python

Exercícios:

2) Crie uma classe Livro que possui os atributos nome, qtdPaginas, autor e preço.

Crie os métodos getPreco para obter o valor do preço e o método setPreco para setar um novo valor do preço.

Crie um código de teste para pelo menos 3 livros.