

Orientação a objeto

Orientação a Objetos surgiu da necessidade de modelar sistemas complexos

- ☐ Modelar problemas utilizando um conjunto de componentes autocontidos, e integráveis
- ☐ Determinar como um objeto deve se comportar e como deve interagir com outros objetos

Algumas iniciativas:

- ☐ Simula 67 (60)
- ☐ Smalltalk (70)
- ☐ C++ (80)

Orientação a objeto

Conceitos essenciais:

- ☐ Classes e objetos
- ☐ Atributos e Métodos
- ☐ Herança
- ☐ Encapsulamento

Orientação a objeto

Os objetos reais possuem duas características:

- ☐ Atributos (Estado)
- ☐ Comportamento

Exemplos:

- ☐ cachorro
 - Atributos: nome, cor, raça
 - Comportamento: latindo, abanando o rabo, comendo

Orientação a objeto

Um objeto de software é conceitualmente similar aos objetos reais

- ☐ Objetos armazenam seu estado em atributos
 - Correspondentes às variáveis em programação estruturada.
- ☐ Objetos expõem seu comportamento através de métodos
 - Correspondentes às funções em programação estruturada.

Orientação a objeto

Exemplos de objeto:

- Gerenciador de Dados de Alunos
 - Atributos: lista de alunos
 - Comportamentos: filtrar alunos por nome, incluir aluno, alterar aluno

- Biblioteca Matemática
 - Atributos: Matriz
 - Comportamentos: calcular transposta, multiplicar, somar

Orientação a objeto

Empacotar o código em objetos individuais fornece:

- ☐ Modularidade
 - Objetos são independente
- ☐ Encapsulamento
 - Os detalhes da implementação de um objeto permanecem ocultos
- ☐ Reuso
 - Objetos podem ser reutilizados em diferentes programas
- ☐ Fraco acoplamento
 - Objetos podem ser substituídos facilmente

Orientação a objeto

Uma classe é o projeto a partir do qual objetos individuais são criados

- Ela define os atributos e os métodos correspondentes aos seus objetos.
- Outros possíveis membros de uma classe são:
 - Construtores: define as operações a serem realizadas quando um objeto é instanciado.
 - Destrutores: define as operações a serem realizadas quando um objeto é destruído.

Orientação a objeto

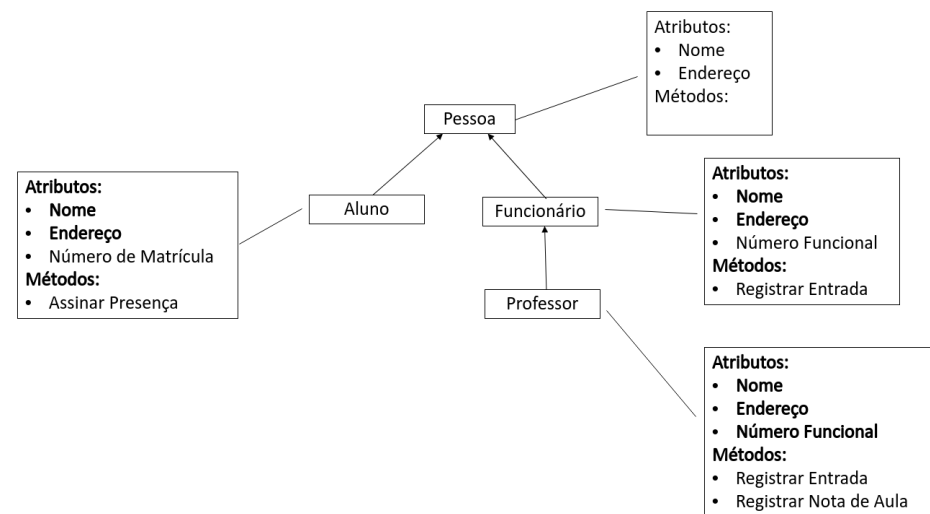
Outras características de uma classe:

- ☐ Uma classe pode herdar características de outra classe e incluir novas características
- ☐ Atributos de uma classe podem ser protegidos, sendo possível alterar seu conteúdo por meio apenas de métodos da própria classe
- ☐ Métodos podem ser reescritos

Orientação a objeto

- ☐ O relacionamento de Herança define um relacionamento do tipo generalização
- ☐ Indica que uma classe (subclasse) é especializada para gerar uma nova (superclasse)
- ☐ Tudo que a superclasse possui, a subclasse também vai possuir
- ☐ Em Python, todas as classes herdam a classe Object

Orientação a objeto



Orientação a objeto

Método Construtor em Python

```
1 def __init__(self):  
2     Comandos do construtor
```

Parâmetro para referenciar ao objeto criado: **self**

- ☐ Para acessar e criar atributos em um objeto, o caracter "." deve ser usado após o nome do objeto.

```
1 class Critter:  
2     def __init__(self, name):  
3         self.name = name
```

Orientação a objeto

A Convenção para definir Métodos ou variáveis privados em Python é colocar como prefixo de seu nome "__"

PS: Não existem elementos privados "verdadeiros" em Python. É possível acessar qualquer método/atributo de uma classe.

- Exemplo:

```
1  __a
2  __my_variable
```

Heranças são definidas na declaração da classe, logo após seu nome

```
1  class teste(object):
2      def __init__(self, X):
3          self.X = X
```

Orientação a objeto

Exemplo de uma classe em Python

```
1 class MyClass:
2     def function(self):
3         print("This is a message inside the class.")
```

Instanciando um objeto e chamando métodos:

```
1 myobjectx = MyClass()
2 myobjectx.function()
```

Orientação a objeto

Exemplo de uma classe em Python

```
1  # Classe que representa uma coordenada X Y
2  class Coordinate(object):
3      #define um construtor
4      def __init__(self, x, y):
5          # configura coordenada x e y
6          self.x = x
7          self.y = y
8      #reimplementa a função __str__
9      def __str__(self):
10         # Representa o em string da coordenada
11         return "<" + str(self.x) + "," + str(self.y) + ">"
```

Orientação a objeto

```
1 def distance(self, other):
2     # Calcula distancia euclidiana entre dois pontos
3     x_diff_sq = (self.x-other.x)**2
4     y_diff_sq = (self.y-other.y)**2
5     return (x_diff_sq + y_diff_sq)**0.5
```

Teste de Uso da Classe

```
1 c = Coordinate(3,4)
2 origin = Coordinate(0,0)
3 print("Coordenada 1:")
4 print(c)
5 print(c.distance(origin))
```

Orientação a objeto

Teste com atributos protegidos

```
1 class MyClass:
2     __variable = 0
3     def setvariable(self, newvar):
4         self.__variable = newvar
5     def getvariable(self):
6         return (self.__variable)
7     def function(self):
8         print("This is a message inside the class.")
```


Orientação a objeto

Teste com atributos protegidos

```
1  var="rs2"  
2  myobjectx = MyClass()  
3  myobjectx.function()  
4  print(myobjectx.getvariable())  
5  var="rs3"  
6  myobjectx.setvariable(var)  
7  print(myobjectx.getvariable())
```

Criando Pacotes para Compartilhamento de Classes

MultiThreading