
# Programação Orientada a Objetos

Revisão

**Paulo Viniccius Vieira**
**paulo.vieira@faculdadeimpacta.com.br**

# Módulos em Python

- Divisão do código em 1 ou mais arquivos
- Vantagens:
  - Organização de código
  - Reaproveitamento de código

- Importação absoluta:

```
import <nome do modulo>
import <nome do modulo> as <nome curto>
```

- Importação relativa (deve ser priorizada):

```
from <nome do modulo> import <funcao1>, <classe1>
```

# Padronização de código PEP8

- Principais regras:
  - Separar classes e funções com 2 linhas em branco
  - Separar métodos da classe com 1 linha em branco
  - Evitar o uso desnecessário de espaços em branco
  - Utilizar apenas 1 espaço em branco quando necessário
  - Limitar tamanho das linhas em 79 caracteres
  - Nomes de variáveis e métodos devem utilizar apenas letras minúsculas
  - Nomes de Classes devem iniciar com letra maiúscula
  - Inserir linha em branco no final do arquivo

# Coleções

- Listas
  - Heterogênea
  - Mutável
  - Delimitada por colchetes [ ]
  - Itens acessados pelo índice

```
lista = [valor1, valor2, valor3]
lista[0] = novo_valor
```

- Tuplas
  - Heterogênea
  - Não Mutável
  - Delimitada por parênteses ( )
  - Itens acessados pelo índice

```
tupla = (valor1, valor2, valor3)
print(tupla[0])
```

# Coleções

- Dicionário
  - Heterogêneo
  - Mutável
  - Delimitado por chaves **{ }**
  - Armazena pares de valores **(chave: valor)**
  - Itens acessados pela chave

```
dic = {chave1: valor1, chave2: valor2}
dic[chave1] = novo_valor
```

# Exceções

- Lançamento de uma exceção com a instrução raise

```
raise TipoDeExcecao
```

- As exceções são classificadas em diferentes tipos

```
(TypeError, ValueError, AssertionError, etc.)
```

- Tratamento de Exceções

  **`try`**: tenta executar um bloco de código.

  **`except`**: caso ocorra uma exceção é executado o bloco except corresponde ao tipo da exceção gerada

  **`else`**: é executado quando não ocorre nenhuma exceção

  **`finally`**: é executado sempre

# Testes Automatizados

- Teste Unitários
  - Testa o retorno de uma função, de acordo com a entrada fornecida
  - Verifica se o resultado está correto através da instrução assert
  - Se a condição for falsa, gera um AssertionError

```
try:
    assert funcao() == valor_esperado
    print('Passou no teste')
except AssertionError:
    print('Ocorreu um erro')
```

# Programação Orientada a Objetos

- É um paradigma de programação, que busca representar entidades do mundo real em objetos computacionais

- Classe:
  - Define a estrutura dos objetos (modelo)
- Objeto:
  - Instância concreta de uma classe
- Atributos:
  - Determinam as características do objeto
- Métodos:
  - Determinam os comportamentos do objeto

# Pilares da POO

- Abstração:
  - representação de entidades do mundo real

- Encapsulamento:
  - protege atributos e métodos de acesso externo
  - atributos e métodos podem ser públicos ou privados
  - atributos privados devem ter o nome iniciado por 2 underlines
  - métodos get e set são utilizados para acessar atributos privados

# Pilares da POO

- Herança:
  - proporciona reaproveitamento de código
  - classe filha herda os atributos e comportamentos da classe mãe
  - herança múltipla ocorre quando uma classe herda de duas ou mais classes

- Polimorfismo:
  - adiciona comportamento dinâmico aos objetos
  - o objeto se comporta de acordo com a classe a qual ele pertence
  - ocorre quando há herança com sobrescrita de métodos

# Classes e métodos abstratos

- Classes Abstratas
  - Não podem ser instanciadas (não geram objetos)
  - Geralmente são utilizadas para definir uma estrutura em comum para as classes filhas

- Métodos Abstratos
  - São métodos que não possuem implementação
  - Só podem existir dentro de classes abstratas
  - É obrigatório que sejam implementados em todas as classes filhas

# Arquivos de Texto

- Três modos básicos de abertura de arquivo:
  - modo **`'r'`** (read) - para ler um arquivo
  - modo **`'w'`** (write) - para criar e escrever em um arquivo
  - modo **`'a'`** (append) - para escrever no final de um arquivo

- Principais funções:
  - **`open()`** - abre o arquivo
  - **`close()`** - fecha o arquivo
  - **`write()`** - escreve uma string no arquivo
  - **`read()`** - copia o conteúdo do arquivo e retorna uma única string
  - **`for`** - repetição for é usada para percorrer as linhas do arquivo

# Conexão com Banco de Dados

- SQLAlchemy Framework
  - A conexão com o banco é feita através da criação de uma *engine* (objeto responsável por gerenciar a troca de mensagens com o banco)
  - A criação da *engine* é feita através de uma url de conexão

- ORM - Object Related Mapping
  - Mapeamento Objeto Relacional
  - Mapeia as tabelas do banco de dados em classes do Python
  - Oferece uma interação mais natural entre a POO e o banco de dados
  - A manipulação dos dados no banco é feita por funções específicas

# Conexão com Banco de Dados

- ORM - Principais funções:

**`add()`** - adiciona os dados de um objeto na tabela

**`add_all()`** - adiciona uma lista de objetos na tabela

**`query()`** - executa uma consulta em uma tabela

**`filter()`** - define filtros a serem utilizados em uma consulta

**`order_by()`** - ordena o resultado de uma consulta por um campo específico da tabela

**`first()`** - retorna o primeiro objeto de uma consulta