
FIAP
MBA

FIAP MBA+

Data Processing With Python

Vagner S. Macedo

Agenda

Aula	Conteúdo Programático	Local
3	Programacao Funcional	Laboratório
	Map	
	Reduce	
	Filter	
	Zip	
	List Comprehension	
	Orientação a Objetos	
	Classe	
	Intuicao	
	Criando uma Classe	
	Herança	

Programação Funcional

Map

- Em Python, a função map() é uma função embutida que permite aplicar uma função a cada item de um iterável (como uma lista, tupla, etc.) e retorna um objeto map que contém os resultados.

- A função map() recebe dois argumentos:

  - Uma função que será aplicada a cada item do iterável.

  - Um ou mais iteráveis (por exemplo, listas, tuplas) cujos elementos serão passados como argumentos para a função.

- A função map() então retorna um objeto map, que é um iterador que produz os resultados da aplicação da função a cada elemento do iterável fornecido.

- Sintaxe:

map(função, iterável1, iteravel2, ...)

Map

- Sintaxe:

```python
# Definindo uma função para dobrar um número
def dobrar(numero):
    return numero * 2

# Criando uma lista de números
numeros = [1, 2, 3, 4, 5]

# Usando a função map para dobrar cada número na lista
resultado = map(dobrar, numeros)

# Convertendo o objeto map em uma lista para visualização dos resultados
lista_resultado = list(resultado)

print(lista_resultado)
```

✓ 0.0s

```
[2, 4, 6, 8, 10]
```

Map

- Se utilizar novamente o objeto map ou iterar, ele zera os valores:

```
# se eu rodar de novo, ele zera
quadrados = map(func, lista)

for n in quadrados:
    print(n)
```

✓ 0.0s

```
4
9
16
```

```
for n in quadrados:
    print(n)
```

✓ 0.0s

- Demonstração

Reduce

- A função **reduce()** perte ao pacote **functools** e aplica uma função a um iterável (como uma lista) de modo cumulativo aos elementos do iterável, de forma que **reduce(func, seq)** funciona do seguinte modo:
  - No **primeiro passo**, os dois primeiros elementos de seq são aplicados à função, por exemplo, **func(seq[0], seq[1])**.
  - No **segundo passo**, a função é chamada com o resultado do primeiro passo e o terceiro elemento de seq, por exemplo, **func(func(seq[0], seq[1]), seq[2])**. Continua assim até que todos os elementos de seq tenham sido aplicados à função.

- Sintaxe:

functools.reduce(funcao, iteravel[, inicial])

funcao: A função a ser aplicada cumulativamente a cada elemento do iterável. Esta função deve aceitar dois argumentos.
iteravel: O iterável (como uma lista) aos quais a função será aplicada cumulativamente.
inicial (opcional): Um valor inicial para ser utilizado no cálculo cumulativo. Se fornecido, a função reduce() retornará esse valor se o iterável estiver vazio.

Reduce

```python
from functools import reduce

# Definindo uma função para calcular a soma cumulativa dos elementos
def soma_cumulativa(a, b):
    return a + b

# Criando uma lista de números
numeros = [1, 2, 3]

# Usando a função reduce para calcular a soma cumulativa dos números na lista
resultado = reduce(soma_cumulativa, numeros)

print(resultado)
```

✓ 0.0s

```
6
```

- Demonstração

Filter

- A função **filter()** é uma função embutida que permite filtrar elementos de um iterável (como uma lista) com base em uma função de filtro. Ela **retorna um iterador** que produz apenas os elementos do iterável para os quais a função de filtro retorna **True**.

- A função filter() recebe dois argumentos:
  - Uma função que retorna True ou False (a função de filtro).
  - Um iterável (como uma lista) que será filtrado.

- Sintaxe:

filter(funcao, iteravel)

Filter

```python
# Definindo uma função de filtro para números pares
def filtro_pares(numero):
    return numero % 2 == 0

# Criando uma lista de números
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Usando a função filter para filtrar apenas os números pares da lista
resultado = filter(filtro_pares, numeros)

# Convertendo o objeto filter em uma lista para visualização dos resultados
numeros_pares = list(resultado)

print(numeros_pares)
```

✓ 0.0s

```
[2, 4, 6, 8, 10]
```

- Demonstração

Zip

- Em Python, a função **zip()** é uma função embutida que combina os elementos de dois ou mais iteráveis (como **listas, tuplas, etc.**) em pares ordenados. Ela retorna um iterador que produz tuplas onde o i-ésimo elemento contém o i-ésimo elemento de cada um dos iteráveis fornecidos.

- A função zip() recebe dois ou mais iteráveis como argumentos e retorna um iterador de tuplas.

- Sintaxe:

zip(iteravel1, iteravel2)

Zip

```python
# Definindo duas listas
nomes = ['Vagner', 'Mariella','Victor']
#nomes = ('Alice', 'Bob', 'Charlie')
idades = (50, 17, 14)

# Usando a função zip para combinar as listas em pares ordenados
pares_ordenados = zip(nomes, idades)

# Convertendo o objeto zip em uma lista para visualização dos resultados
lista_pares = list(pares_ordenados)

print(lista_pares)
```

✓ 0.0s

```
[('Vagner', 50), ('Mariella', 17), ('Victor', 14)]
```

- Demonstração

List Comprehension

- Uma **list comprehension** é uma forma concisa de **criar listas**.
- Permite criar listas usando uma única linha de código de maneira mais legível e expressiva.
- A list comprehension consiste em uma expressão seguida por uma **cláusula for** e, opcionalmente, **cláusulas if para filtrar os itens**.
- Ela **retorna uma nova lista** resultante da avaliação da expressão para cada item do iterável.
- Sintaxe:

[expressão for item in iterável]

ou

[expressão for item in iterável if condição]

List Comprehension

1. List Comprehension **simples**:

```python
# com condição
numeros = [1, 2, 3, 4, 5]
quadrados = [numero ** 2 for numero in numeros]
print(quadrados)  # Output: [1, 4, 9, 16, 25]
```

✓ 0.0s

```
[1, 4, 9, 16, 25]
```

List Comprehension

2. List Comprehension **com condição**:

```python
# simples
numeros = [1, 2, 3, 4, 5]
quadrados_pares = [numero ** 2 for numero in numeros if numero % 2 == 0]
print(quadrados_pares)  # Output: [4, 16]
```

✓ 0.0s

```
[4, 16]
```

List Comprehension

3. List Comprehension **com Strings**:

```python
# com strings
palavras = ['python', 'é', 'show']
maiusculas = [palavra.upper() for palavra in palavras]
print(maiusculas)  # Output: ['PYTHON', 'É', 'LEGAL']
```

✓ 0.0s

```
['PYTHON', 'É', 'SHOW']
```

List Comprehension

4. List Comprehension **aninhada**:

```python
# aninhada
matriz = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
achatada = [numero for linha in matriz for numero in linha]
print(achatada)  # Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

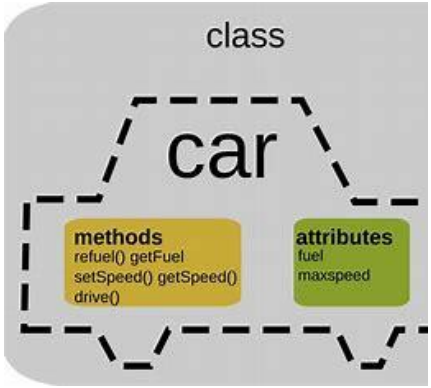
✓ 0.0s

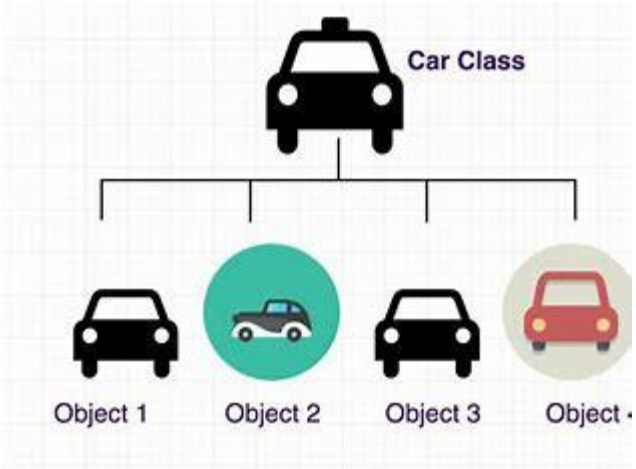
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Demonstração

Orientação a Objetos

Classe – Orientação a Objetos





Classe – Definição

- Uma classe em Python é uma **estrutura (molde)** que encapsula dados (**atributos**) e comportamentos (**métodos**) relacionados. Ela define um tipo de objeto e como esse objeto pode ser criado e interagir com outros objetos.

- As classes permitem organizar e **reutilizar código** de forma eficiente, promovendo a **modularidade** e a **legibilidade do código**.

- Componentes de uma Classe:

  - Atributos: **Variáveis** que armazenam dados associados a um objeto.

  - Métodos: **Funções** que definem o **comportamento** dos objetos da classe.

  - Construtor: Um método especial chamado **__init__()** que é **chamado automaticamente** quando um objeto da classe é **criado**. Ele **inicializa** os atributos do objeto.

  - Self: Uma **referência** ao **próprio** objeto **dentro** dos métodos da classe. É usado para **acessar os atributos** e **métodos** do objeto.

Classe - Exemplo

```python
class Carro:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def dirigir(self):
        print(f"O {self.marca} {self.modelo} está sendo dirigido.")

    def parar(self):
        print(f"O {self.marca} {self.modelo} parou.")

# Criando um objeto da classe Carro
meu_carro = Carro("Toyota", "Corolla")

# Chamando métodos do objeto
meu_carro.dirigir()  # Output: O Toyota Corolla está sendo dirigido.
meu_carro.parar()    # Output: O Toyota Corolla parou.
```

✓ 0.0s

```
O Toyota Corolla está sendo dirigido.
O Toyota Corolla parou.
```

Classe – Outro Exemplo

```python
class Calculadora:
    def __init__(self):
        pass

    def somar(self, a, b):
        return a + b

    def subtrair(self, a, b):
        return a - b

    def multiplicar(self, a, b):
        return a * b

    def dividir(self, a, b):
        if b == 0:
            return "Erro: divisão por zero!"
        else:
            return a / b

# Criando uma instância da classe Calculadora
minha_calculadora = Calculadora()

# Realizando operações matemáticas
print("2 + 3 =", minha_calculadora.somar(2, 3))
print("5 - 1 =", minha_calculadora.subtrair(5, 1))
print("4 * 6 =", minha_calculadora.multiplicar(4, 6))
print("10 / 2 =", minha_calculadora.dividir(10, 2))
print("8 / 0 =", minha_calculadora.dividir(8, 0))
```

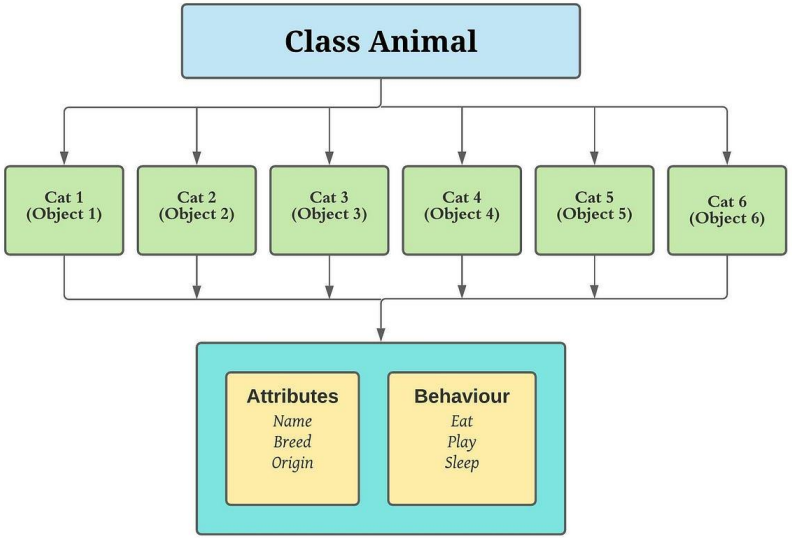
✓ 0.0s

```
2 + 3 = 5
5 - 1 = 4
4 * 6 = 24
10 / 2 = 5.0
8 / 0 = Erro: divisão por zero!
```

Classe - Herança

- Herança em Python é a capacidade de uma classe **herdar características (atributos e métodos)** de outra classe.

- A classe que está sendo herdada é chamada de **superclasse** ou classe base, e a classe que herda é chamada de **subclasse**.



Herança - Exemplo

```python
class Funcionario:
    def __init__(self, nome, salario):
        self.nome = nome
        self.salario = salario

    def calcular_pagamento(self):
        return self.salario

class Gerente(Funcionario):
    def __init__(self, nome, salario, bonus):
        super().__init__(nome, salario)
        self.bonus = bonus

    def calcular_pagamento(self):
        return self.salario + self.bonus

# Criando instâncias de Funcionario e Gerente
funcionario1 = Funcionario("João", 3000)
gerente1 = Gerente("Maria", 5000, 1000)

# Calculando pagamento para cada funcionário
print("Pagamento de", funcionario1.nome + ":", funcionario1.calcular_pagamento())  # Output: Pagamento de João: 3000
print("Pagamento de", gerente1.nome + ":", gerente1.calcular_pagamento())          # Output: Pagamento de Maria: 6000
```

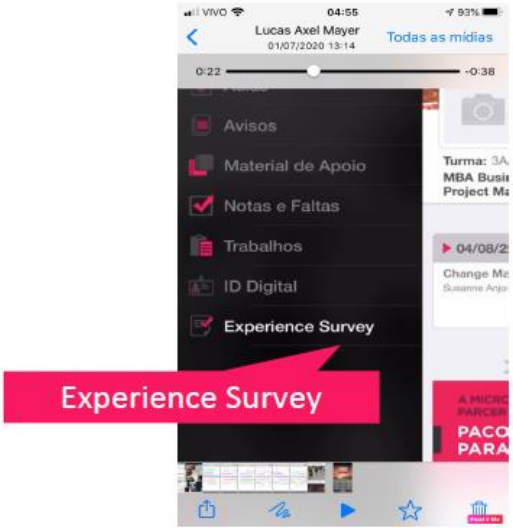
✓ 0.0s

```
Pagamento de João: 3000
Pagamento de Maria: 6000
```

- Demonstração

O que você achou da aula de hoje?

Entrar no aplicativo FIAPP, e no menu clicar em Experience Survey



Obrigado!

/vagner-dos-santos

profvagner.macedo@fiap.com.br


FIAP