# Engenharia de Software
# Computacional thinking with Python
# Aula 12 – Checkpoint 2

Prof. Dr. Francisco Elânio

# Exercício 1

**Problema: Cálculo de Desempenho de Funcionários**

**Descrição: Você é responsável por calcular o desempenho dos funcionários com base em suas avaliações trimestrais. Cada funcionário tem uma lista de notas, e você precisa calcular a média de desempenho.**

**Tarefa: Escreva uma função que receba uma lista de notas e retorne a média. Use uma expressão lambda para filtrar quaisquer notas inválidas (por exemplo, notas negativas), além de mostrar a média.**

```
notas_funcionario1 = [8, 7, 6, -2, 9]
notas_funcionario2 = [6, 5, 7, 8, 7]
```

# Exercício 1

```python
def calcular_desempenho(notas):
    is_nota_valida = lambda nota: nota >= 0
    total_notas_validas = 0
    soma_notas_validas = 0

    for nota in notas:
        if is_nota_valida(nota):
            soma_notas_validas += nota
            total_notas_validas += 1

    if total_notas_validas > 0:
        media_desempenho = soma_notas_validas / total_notas_validas
        print("A média de desempenho é:", media_desempenho)
        return media_desempenho
    else:
        print("Não há notas válidas para calcular a média.")
        return None

notas_funcionario1 = [8, 7, 6, -2, 9]
notas_funcionario2 = [6, 5, 7, 8, 7]

print("Funcionário 1:")
media_funcionario1 = calcular_desempenho(notas_funcionario1)
print("\nFuncionário 2:")
media_funcionario2 = calcular_desempenho(notas_funcionario2)
```

# Exercício 2

**Problema: Classificação de Produtos**

**Descrição: Você tem uma lista de produtos com seus preços e categorias. Precisa classificá-los por categoria e calcular a média de preço para cada categoria.**

**Tarefa: Escreva uma função que receba uma lista de produtos, onde cada produto é representado por um dicionário com as chaves 'nome', 'categoria' e 'preço'. Use uma expressão lambda para agrupar os produtos por categoria e calcule a média de preço para cada uma.**

```
    {'nome': 'Camiseta', 'categoria': 'Roupas', 'preco': 29.99},
    {'nome': 'Calça', 'categoria': 'Roupas', 'preco': 49.99},
    {'nome': 'Tênis', 'categoria': 'Calçados', 'preco': 99.99},
    {'nome': 'Meia', 'categoria': 'Acessórios', 'preco': 9.99},
    {'nome': 'Boné', 'categoria': 'Acessórios', 'preco': 19.99}
```

# Exercício 2

```python
def classificar_produtos(produtos):
    agrupar_por_categoria = lambda produto: produto['categoria']
    produtos_por_categoria = {}

    for produto in produtos:
        categoria = agrupar_por_categoria(produto)
        if categoria in produtos_por_categoria:
            produtos_por_categoria[categoria].append(produto)
        else:
            produtos_por_categoria[categoria] = [produto]

    media_precos_por_categoria = {}
    for categoria, produtos_categoria in produtos_por_categoria.items():
        precos = [produto['preco'] for produto in produtos_categoria]
        media_precos_por_categoria[categoria] = sum(precos) / len(precos)

    return media_precos_por_categoria

produtos = [
    {'nome': 'Camiseta', 'categoria': 'Roupas', 'preco': 29.99},
    {'nome': 'Calça', 'categoria': 'Roupas', 'preco': 49.99},
    {'nome': 'Tênis', 'categoria': 'Calçados', 'preco': 99.99},
    {'nome': 'Meia', 'categoria': 'Acessórios', 'preco': 9.99},
    {'nome': 'Boné', 'categoria': 'Acessórios', 'preco': 19.99}
]
media_precos_por_categoria = classificar_produtos(produtos)
print("Média de preço para cada categoria:", media_precos_por_categoria)
```

# Exercício 3

**Problema: Análise de Texto**

**Descrição: Você recebeu um grande conjunto de textos e precisa contar a frequência de cada palavra. Além disso, deseja remover palavras comuns, como** `"o", "a", "de", "e", "para", "com"].`

**Tarefa: Escreva uma função que receba uma lista de textos e retorne um dicionário com a frequência de cada palavra. Use a função def para resolver o problema.**

```
Frases a serem analisadas:

"O tempo está agradável para passear e curtir.",
"O parque é um ótimo lugar para curtir.",
"As pessoas gostam de passear no parque com o tempo agradável."
```

# Exercício 3

```python
def analisar_texto(textos):
    frequencia_palavras = {}

    palavras_comuns = ["o", "a", "de", "e", "para", "com"]

    for texto in textos:
        palavras = texto.split()
        for palavra in palavras:
            palavra_normalizada = palavra.lower()
            if palavra_normalizada not in palavras_comuns:
                if palavra_normalizada in frequencia_palavras:
                    frequencia_palavras[palavra_normalizada] += 1
                else:
                    frequencia_palavras[palavra_normalizada] = 1

    return frequencia_palavras

textos = [
    "O tempo está agradável para passear e curtir.",
    "O parque é um ótimo lugar para curtir.",
    "As pessoas gostam de passear no parque com o tempo agradável.",
]
frequencia_palavras = analisar_texto(textos)
print("Frequência de cada palavra:", frequencia_palavras)
```

# Exercício 4

**Problema: Simulação de Estoque**

**Descrição: Você está desenvolvendo um sistema de simulação de estoque para uma empresa. Precisa simular o comportamento do estoque ao longo do tempo com base em vendas e reposições.**

**Tarefa: Escreva uma função que simule o estoque ao longo de um período de tempo, recebendo listas de vendas e reposições como entrada. Use uma função definida pelo usuário para calcular o novo nível de estoque com base nas transações.**

| | | | | |
|---|---|---|---|---|
| **vendas** | 10 | 15 | 8 | 12 |
| **reposicoes** | 5 | 7 | 6 | 5 |
| **estorque_inicial** | 100 | | | |

# Exercício 4 - Resolução

```python
def simular_estoque(vendas, reposicoes, estoque_inicial, calcular_novo_estoque):
    """Simula o estoque ao longo do tempo com base em vendas e reposições."""
    estoque = estoque_inicial
    historico_estoque = [estoque]

    for venda, reposicao in zip(vendas, reposicoes):
        estoque = calcular_novo_estoque(estoque, venda, reposicao)
        historico_estoque.append(estoque)

    return historico_estoque

def calcular_novo_estoque(estoque_atual, venda, reposicao):
    """Calcula o novo nível de estoque com base nas vendas e reposições."""
    novo_estoque = estoque_atual - venda + reposicao
    return max(novo_estoque, 0)

vendas = [10, 15, 8, 12]
reposicoes = [5, 7, 6, 5]
estoque_inicial = 100

historico_estoque = simular_estoque(vendas, reposicoes, estoque_inicial,
calcular_novo_estoque)
print("Histórico de estoque ao longo do tempo:", historico_estoque)
```

# Exercício 5

**Problema: Estimativa de Despesas**

**Descrição: Você precisa estimar as despesas mensais de uma empresa com base em diferentes categorias, como salários, aluguel, fornecedores. Cada categoria tem uma lista de valores mensais.**

**Tarefa: Escreva uma função que receba um dicionário de categorias, onde cada categoria é uma chave com uma lista de valores. Use uma expressão lambda para calcular a média de despesas em cada categoria ao longo do tempo**

| | | | | |
|---|---|---|---|---|
| **salarios** | 5000 | 5500 | 6000 | 5800 |
| **aluguel** | 1500 | 1500 | 1600 | 1600 |
| **fornecedores** | 2000 | 2200 | 2300 | 2100 |

# Exercício 5 - Resolução

```
def estimar_despesas(categorias):
    media_despesas = {categoria: sum(valores) /
len(valores) for categoria, valores in
categorias.items()}
    return media_despesas

categorias = {
    'Salarios': [5000, 5500, 6000, 5800],
    'Aluguel': [1500, 1500, 1600, 1600],
    'Fornecedores': [2000, 2200, 2300, 2100]
}
media_despesas = estimar_despesas(categorias)
print(media_despesas)
```

# Exercício 6

```
def calcular_raizes_quadraticas(a, b, c):
    discriminante = b**2 - 4*a*c

    calcular_x1_x2 = lambda a, b, c: (
        (-b + discriminante**0.5) / (2*a),
        (-b - discriminante**0.5) / (2*a)
    )

    return calcular_x1_x2(a, b, c)

raizes = calcular_raizes_quadraticas(1, -3, 2)
print("Raiz x1:", raizes[0])
print("Raiz x2:", raizes[1])
```

**Supondo que a função calcular_raizes_quadraticas seja usada repetidamente em um programa para resolver várias equações quadráticas com os mesmos coeficientes A, B e C, como você poderia modificar a função para evitar recálculos desnecessários do discriminante?**

**Obs: Mostre como modificar a função para calcular o discriminante apenas uma vez e armazená-lo em uma variável, que pode ser reutilizada nas chamadas subsequentes da função.**

# Exercício 6

```
def calcular_raizes_quadraticas(a, b, c):
    discriminante = b**2 - 4*a*c

    if a == 0:
        raiz_linear = -c / b
        return raiz_linear
    else:
        if discriminante >= 0:
            calcular_x1_x2 = lambda a, b, c, discriminante: (
                (-b + discriminante**0.5) / (2*a),
                (-b - discriminante**0.5) / (2*a)
            )
            return calcular_x1_x2(a, b, c, discriminante)
        else:
            return "Raízes complexas, discriminante negativo"

raizes = calcular_raizes_quadraticas(1, -3, 2)
print("Raiz x1:", raizes[0])
```

# Exercício 7

Leia as afirmações abaixo sobre função lambda, apply, def e marque se afirmação e/ou código apresentado esta de acordo com a afirmação de cada questão. Marque verdadeiro ou falso.

1. Ao utilizar uma função lambda, podemos reduzir a necessidade de definir funções separadas, especialmente para operações simples.

```
import pandas as pd
df['nova_coluna'] = df['coluna_existente'].apply(lambda x: x * 2)
```

2. Podemos usar a função apply em um DataFrame para aplicar uma função personalizada que aceita múltiplos argumentos.

```
import pandas as pd
def minha_funcao(x, y):
    return x + y
df['nova_coluna'] = df.apply(lambda row: minha_funcao(row['coluna1'], row['coluna2']), axis=1)
```

3. Leia o código abaixo com função def, lambda, return e descreva que a função está correta e o código funciona perfeitamente.

```
def outer_function():
    inner_lambda = lambda x: x * 2
    return inner_lambda(5)
print(outer_function(5))
```

4. As funções lambda em Python são nomeadas e podem ser chamadas em qualquer lugar do código.

5. A função apply do pandas aplica a função especificada a cada linha ou coluna de um DataFrame, não a cada elemento de uma série.

# Exercício 7

a) Verdadeiro

b) Verdadeiro

c) Falso

d) Falso

e) Falso

*“O que sabemos é uma gota; o que ignoramos é um oceano.”*
*(Issac Newton)*

# Referências


- ASCENCIO, A. F. G, CAMPOS, E. A. V. Fundamentos da Programação de Computadores: algoritmos, Pascal, C/C++ e Java, 2ª Edição, São Paulo: Pearson 2007.
- FURGERI, Sérgio. Introdução à Programação em Python. São Paulo: Editora Senac, 2021.
- MENEZES, Nilo. Introdução à Programação em Python. São Paulo: Novatec, 2019
- SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madson. Algoritmos. São Paulo: Pearson, 2004.