Atividade S2 A4

Nome: Adalberto Caldeira Brant Filho

Repositório GitHub: https://github.com/adalbertobrant/lipai

Código das Videoaulas

Aula 01 - Funções

Funções são blocos de códigos, modularizáveis que podem ser reutilizadas, realizando tarefas específicas.
As funções evitam a duplicação de código
O python já tem algumas funções que são conhecidas como Standard Library Functions ou built-in functions, essas funções já foram definidas, sendo apenas chamadas
ex : print, len
As funções podem ter parametros ou não:

```
def oi(nome):
    print(f'Oi, {nome}')
```

```
  """" Aula 01 - Introdução a funções """

# funções built-in functions

nome = 'José'
print(nome)
print(len(nome))

# User Defined Functions
# são definidas pelo usuário do programa python
# fazem parte da solução do problema

# função saudacoes

def saudacoes():
    print("Hello World")


# para imprimir o resultado da função saudacoes, deve-se chamar a função
saudacoes()
saudacoes()

# imprime saudacao com nome

def ola(nome):
    print(f'Olá, {nome}')


# maria é o argumento do parametro nome
```

```
ola('Maria')

# função para calcular a media

def calcular_media(nota1, nota2, nota3):
    media = (nota1 + nota2 + nota3)/3
    print(media)


# chamada
# argumentos são literais
calcular_media(10, 5, 7)

n1 = 5
n2 = 7
n3 = 8

# chamada
# argumentos são variáveis
calcular_media(n1, n2, n3)

# Declaração
# nome: media
# parametros: nota1, nota2, nota3
# retorno: media

def media(nota1, nota2, nota3):
    return (nota1 + nota2 + nota3)/3.0

media = media(10, 7, 4)
print(f'Valor da média é {media}')

# quando colocamos um retorno na função ele pode ser usado em diferentes
cenários, como salvar em um banco de dados, enviar um email , salvar em um
arquivo
```

Aula 02 -

```
""" Aula 02 - Arguments: positional , keyword, default value """

# declara uma função que soma dois números
def somar(n1, n2):
    #   soma = n1 + n2
    #   return soma
    return n1 + n2


# forma de chamar é argumentos posicionais
# argumentos vão de acordo com a posição
print(somar(3.4, 10))

# argumentos nomeados onde o parametro é nomeado com o seu literal
print(somar(n1=5, n2=10))
```

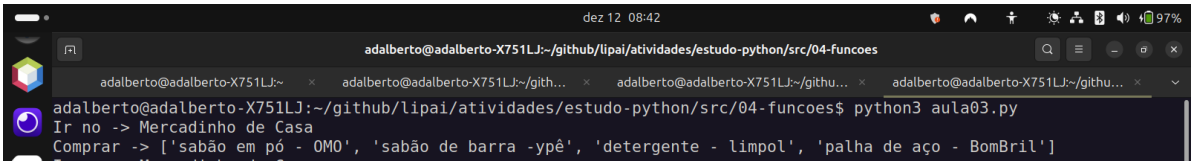
```
print(somar(n2=10, n1=5))

# função dividir
def dividir(dividendo, divisor):
    if divisor != 0:
        return dividendo / divisor
    return "Erro de argumento da função"

# argumentos posicionais
print(dividir(10, 2))
print(dividir(divisor=3, dividendo=10))
print(dividir(dividendo=10, divisor=2))
print(dividir(*[0, 0]))

# fazendo unpack de listas ou tuplas e enviando para a função
numeros = [10, 32]
print(f'Somar números de uma lista -> {somar(numeros[0], numeros[1])}')

# *args(lista)

print(f'Somar números de uma lista -> {somar(*numeros)}')
tuplas_numeros = (11, -4)
print(f'Somando tuplas -> {somar(*tuplas_numeros)}')

""" Ao fazer o unpack devemos ter o cuidado que a
coleção enviada deve ter o mesmo tamanho dos argumentos da função """

# unpack de dicionários **kwargs
numeros = {
    'n1': 100.45,
    'n2': 35.86
}
print(f'Soma de dois valores do dicionário -> {somar(**numeros)}')
""" Ao enviar por dicionário ele usa os valores das chaves do dicionário
que tem que ser o mesmo nome dos argumentos que já estão na função """

# valores padrões em uma função
# Declaração
# nome: saudacoes
# parametros: nome-> deve passar obrigatoriamente, saudacao-> terá um valor
default
# retorno: string
def saudacoes(nome, saudacao='Oi'):
    return f'{saudacao} {nome} !'


print(saudacoes('João', 'Olá'))
print(saudacoes('Pedro'))
```

Aula 04 - *args, **kwargs

Os *args são listas e os **kwargs são dicionários, eles podem ser úteis dentro dos parametros de uma função, ajudando a diminuir a quantidade de termos da mesma.
Em vez de passarmos N argumentos podemos utilizar uma lista de *args

```python
""" Aula 03 - Python: entendendo o uso de *args e **kwargs em funções e métodos
"""

# *args
# o nome args vem de argumentos no plural pois podem ser vários
# o *args funciona como uma lista
def compras(nome_supermercado, *lista_compras):
    print(f'Ir no -> {nome_supermercado}')
    for produtos in lista_compras:
        print(f'Comprar -> {produtos}')


super = 'Mercadinho de Casa'
comprar = ['sabão em pó - OMO',
           'sabão de barra -ypê',
           'detergente - limpol',
           'palha de aço - BomBril',
           ]
# ao passar os dois argumentos da função sem usar o *comprar
# observamos que a lista comprar é entendida como apenas um único elemento
compras(super, comprar)
```

No código vemos que a lista comprar foi entendida pela função como tendo apenas um único argumentos, observer a saída na figura abaixo



Vamos agora passar a lista comprar utilizando a forma correta e esperada na função que é:

```python
# passar a lista comprar utilizando a forma *args -> *comprar
# o programa funciona da forma esperada
compras(super, *comprar)
```

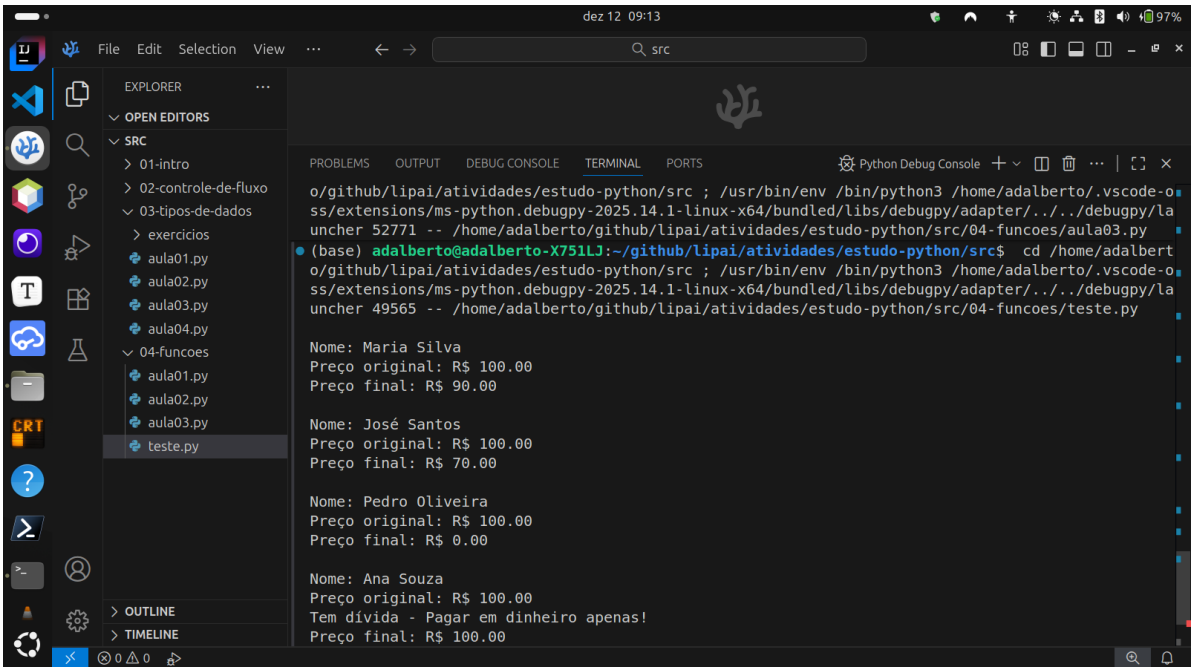
Podemos também passar para um função em python argumentos do tipo dicionário chamados de keyword arguments ou **kwargs

```python
 # **kwargs
# O **kwargs no python é conhecido como keyword arguments, ou seja é necessário
# passar uma chave e um valor para utilizar esse argumento
```

```python
# Dada a situação de que clientes tem desconto de 10% , funcionários tem desconto
de 30%
# devedores não tem desconto e devem pagar apenas no dinheiro
# e o dono leva de graça os produtos do supermercado
# como criar o **kwards dessa situação ?

def calcular_preco(preco, **kwargs):
    descontos = {
        'cliente': 0.10,
        'funcionario': 0.30,
        'dono': 1.0
    }
    desconto = 0

    for tipo, valor in kwargs.items():
        if tipo in descontos:
            desconto = descontos[tipo]
            break

    # Calcular o preço final com desconto
    preco_final = preco * (1 - desconto)

    # Verificar condição de pagamento para devedores
    if kwargs.get('devedor', False):
        print("Tem dívida - Pagar em dinheiro apenas!")
        return preco  # Sem desconto para devedores

    return preco_final


# Dicionário de pessoas
pessoas = {
    'maria': {
        'nome': 'Maria Silva',
        'idade': 35,
        'cliente': True
    },
    'jose': {
        'nome': 'José Santos',
        'idade': 45,
        'funcionario': True
    },
    'pedro': {
        'nome': 'Pedro Oliveira',
        'idade': 50,
        'dono': True
    },
    'ana': {
        'nome': 'Ana Souza',
        'idade': 28,
        'devedor': True
    }
}

preco_produto = 100.0
```

```python
for pessoa, detalhes in pessoas.items():
    print(f"\nNome: {detalhes['nome']}")
    print(f"Preço original: R$ {preco_produto:.2f}")

    # Passar os kwargs dinamicamente
    preco_final = calcular_preco(
        preco_produto, **{k: v for k, v in detalhes.items()
                          if k != 'nome' and k != 'idade'})

    print(f"Preço final: R$ {preco_final:.2f}")
```

A foto mostra a execução do programa exemplo



Cada cliente tem o seu desconto e o dono leva de graça o produto, devedores apenas pagam em dinheiro.

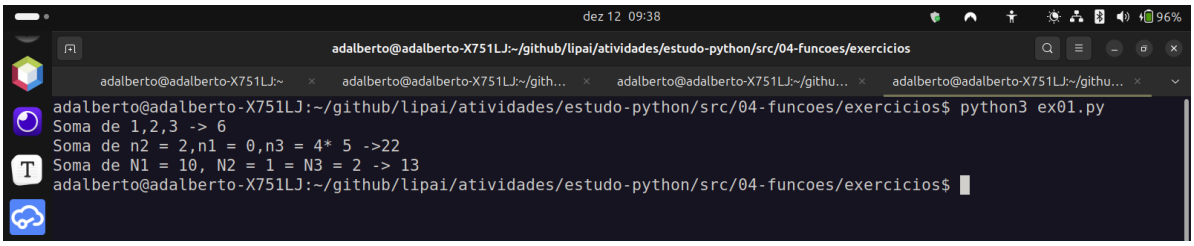
Exercícios de fixação

Ex01

```python
""" ex01.py: Crie uma função que recebe três números
como parâmetro (n1, n2,n3) e imprime na saída padrão a soma dos números."""

# declaração: função soma
# recebe 3 números como parâmetros
# imprime na saída padrão não tem retorno a função


def soma(n1, n2, n3):
    """ soma de 3 números """
    print(n1 + n2 + n3, sep="")


print('Soma de 1,2,3 -> ', end="")
soma(1, 2, 3)
```

```
print('Soma de n2 = 2,n1 = 0,n3 = 4* 5 ->', end="")
soma(n2=2, n1=0, n3=4*5)

N1 = 10
N2 = 1
N3 = 2

print('Soma de N1 = 10, N2 = 1 = N3 = 2 -> ', end="")
soma(N1, N2, N3)
```

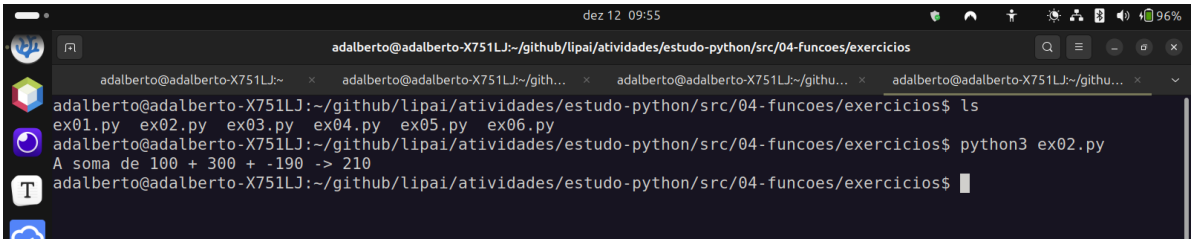
Demonstração do exercício 01



Ex02

```
""" ex02.py: Crie uma função que recebe três números como parâmetro (n1, n2,
n3) e retorna a soma dos números."""


def somar_numeros(n1=0, n2=0, n3=0):
    """ retorna a soma dos números n1,n2,n3 """
    return n1+n2+n3


N1 = 100
N2 = 300
N3 = -190

print(f'A soma de {N1} + {N2} + {N3} -> {somar_numeros(n2=N2, n1=N1, n3=N3)}')
```

Demonstração do exercício 02



Ex03

```
""" ex03.py: Crie uma função que recebe uma tupla de números como
parâmetro (numeros) e retorna a soma desses números """


def soma_tuplas(tuplas_numeros=()):
    """ retorna a soma dos elementos da tupla """
```

```
    somador_tupla = 0
    for n in tuplas_numeros:
        somador_tupla += n

    return somador_tupla


tuplas = (1, 2, 3, 4, 5, 6, 9, 100, -23, -45, 10, -34)

print(
    f'  A soma dos números na tupla -> {tuplas}\n  É de {soma_tuplas(tuplas)}')
```

Ex04

```
""" ex04.py: Crie uma função que recebe vários argumentos numéricos através
de *args e retorna a soma dos números. """


def soma_args(*args):
    """ retorna a soma de uma lista de números """
    soma = 0
    for n in args:
        soma += n
    return soma


lista_numeros = [1.1, 2.5, 6.7, 3, 45, 100, -93,
                 98, -100.95, 33, 52.99, -100.76, 30]

print(
    f'A soma da lista de números -> {lista_numeros}\nÉ
{soma_args(*lista_numeros)}')
```

Ex05

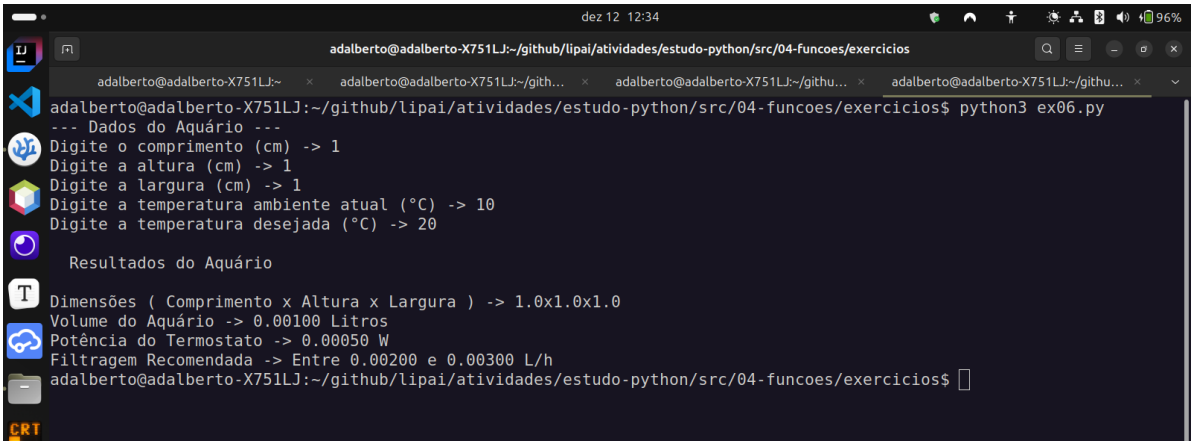
```
""" ex05.py: Implemente uma calculadora de Índice de Massa Corporal (IMC)
usando as diretrizes da OMS. Restante do enunciado na folha de exercícios
entregue no teams
"""
```

```python
def calcular_imc(individuo):
    """Retorna o IMC de um indivíduo com base na sua altura e peso."""
    altura, peso = individuo.values()
    return peso / (altura * altura)

def obter_classificacao(imc):
    """Retorna a classificação com base no IMC."""
    classificacao_imc = {
        (0.0, 18.4, 'Baixo peso'),
        (18.5, 24.9, 'Peso normal'),
        (25.0, 29.9, 'Excesso de peso'),
        (30.0, 34.9, 'Obesidade de Classe 1'),
        (35.0, 39.9, 'Obesidade de Classe 2'),
        (40.0, 999999999, 'Obesidade de Classe 3'),
    }
    for valor_inicial, valor_final, classificacao in classificacao_imc:
        if valor_inicial <= imc <= valor_final:
            return classificacao
    return None

def situacao_individuo(imc):
    """Retorna 'normal', 'perder peso' ou 'ganhar peso' usando dicionário de
intervalos."""
    pesos = {
        (0.0, 18.4): 'ganhar peso',
        (18.5, 24.9): 'normal',
        (25.0, 999999999): 'perder peso',
    }

    for (minimo, maximo), indicacao in pesos.items():
        if minimo <= imc <= maximo:
            return indicacao
    return None

# Entrada de dados pelo usuário
altura = float(input('Digite a sua altura -> '))
peso = float(input('Digite o seu peso atual -> '))

pessoa = {
    'altura': altura,
    'peso': peso
}

imc = calcular_imc(pessoa)
classificacao_imc = obter_classificacao(imc)
situacao = situacao_individuo(imc)

print("  Calculadora IMC  ")
print()
print()
print(f'IMC = {pessoa['peso']} / {pessoa['altura']}²')
print(f'IMC -> {imc:.2f}')
print(f'Classificação IMC -> {classificacao_imc}')
print(f'Recomendações -> {situacao}')
```

Ex 06

```
""" ex06.py: Crie um programa em Python que recebe como entrada o
comprimento, altura e largura (em cm) de um aquário e calcule:
○ O volume do aquário em litros;
○ A potência do termostato necessária para manter a temperatura adequada;
○ A quantidade de filtragem por hora (em litros) necessária para manter a
qualidade da água.
"""


def calcular_volume(medidas):
    """Calcula o volume do aquário em litros com base nas dimensões."""
    comp = medidas['comprimento']
    alt = medidas['altura']
    larg = medidas['largura']

    return (comp * alt * larg) / 1000


def calcular_potencia_termostato(volume, dados_aquario):
    """Calcula a potência do termostato baseada no volume e temperaturas."""
    temp_desejada = dados_aquario['temp_desejada']
    temp_ambiente = dados_aquario['temp_ambiente']

    return volume * 0.05 * (temp_desejada - temp_ambiente)


def calcular_filtragem(volume):
    """Retorna uma tupla com a filtragem mínima e máxima (2 a 3 vezes o
volume)."""
    minima = volume * 2
    maxima = volume * 3
    return minima, maxima


# Entrada de dados pelo usuário
print("--- Dados do Aquário ---")
comprimento = float(input('Digite o comprimento (cm) -> '))
altura = float(input('Digite a altura (cm) -> '))
largura = float(input('Digite a largura (cm) -> '))
temp_ambiente = float(input('Digite a temperatura ambiente atual (°C) -> '))
temp_desejada = float(input('Digite a temperatura desejada (°C) -> '))
```

```
aquario = {
    'comprimento': comprimento,
    'altura': altura,
    'largura': largura,
    'temp_ambiente': temp_ambiente,
    'temp_desejada': temp_desejada
}

volume_litros = calcular_volume(aquario)
potencia = calcular_potencia_termostato(volume_litros, aquario)
filtragem_min, filtragem_max = calcular_filtragem(volume_litros)


print("\n  Resultados do Aquário  ")
print()
print(f'Dimensões ( Comprimento x Altura x Largura ) ->
{aquario["comprimento"]}x{aquario["altura"]}x{aquario["largura"]}')
print(f'Volume do Aquário -> {volume_litros:.5f} Litros')
print(f'Potência do Termostato -> {potencia:.5f} W')
print(f'Filtragem Recomendada -> Entre {filtragem_min:.5f} e {filtragem_max:.5f}
L/h')
```



Reflexão sobre Funções

Qual função é mais flexível em relação ao uso: a do ex01 (que imprime) ou a do ex02 (que retorna o valor)? Por quê?

R: A função mais flexível é a do exercício 02 pois pode-se reaproveitar o código em diferentes cenários, tais como retornar o valor para outra função que pega uma função de soma e divida por dois por exemplo. Isso seria impossível utilizando a função do exercício 01.

Qual abordagem é mais flexível: a do ex02 (3 parâmetros fixos) ou a do ex03/ex04 (que permitem número variável de argumentos)?

R: A utilização de uma função com argumentos variáveis tem a sua utilidade mas depende muito do contexto em que se quer utilizar a função , entretanto de maneira geral podemos considerar que uma abordagem mais flexível como apresentada nos ex03 e ex04 é preferencial no caso de n parâmetros diferentes e que algumas vezes o usuário nem sabe quais são.

As funções do ex03 e ex04 permitem enviar um número variável de parâmetros (tupla ou *args). Em que situações você prefere cada forma? Justifique.

R: Quando o usuário tem liberdade para passar os parâmetros, acredito que seja mais indicado a utilização das listas(*args), caso sejam de tipos iguais e possam ser utilizados para apresentação de dados e valores, por serem mais semânticas e ter mais conveniência. Já no caso de passagem de tuplas acredito que seja mais em processamento de datasets que necessitam de uma estrutura mais rígida.