Mundo 3 - Funções 2

1) Calcule o fatorial de um número e mostre na tela o resultado

a)

Esta foi a solução que eu encontrei e ele funciona da seguinte forma:

- Do lado global do código temos n valendo um input pedindo ao usuário um número inteiro
- Após isso chamamos uma def chamada fatorial com o n levado a parâmetro

Entendendo a def parâmetro:

- chamamos ela de fatorial passando atribuindo num como receptor de parâmetro
- variável local total recebe valor de num
- enquanto num for maior que 1, então
  - print de num x com o final de '' para juntar as linhas
  - num = num - 1
- total = total * num
- Quando num for maior ou igual a 1 então ele vai printar 1 = total

Esta foi uma das formas que eu fiz com o while, mas tem como fazer com o for

```
# fatorial = 5! = 5 x 4 x 3 x 2 x 1 = 120

def fatorial(num):
  total = num
  print(f'{num}! = ',end='')
  while num > 1:
    print(f'{num} x ',end = '')
    num -= 1
    total *= num
  print(f'1 = {total}')

n = int(input('Numero: '))
fatorial(n)
```

b)

Esta é a forma com o for in range, a sua estrutura é bem semelhante à anterior, sendo a diferença o tamanho e a necessidade de colocarmos condicionais para os prints.

- Então definimos no global a chamada da def fatorial pelo n recebendo um input de número inteiro
- A função fatorial recebe um parâmetro opcional se não for digitada será considerada o valor de 1
- a primeira linha é um print com o valor digitado informando que será o fatorial igual

- para c in range de (inicio N, final 0 (para ir até 1), passo -1)
- se c for maior que 1 ele printa c x
- se c for igual a 1 ele printa só o c
- final = final * c
- print do valor final
- então o resultado ficaria 5! = 5 × 4 × 3 × 2 × 1 = 120

```
def fatorial (n=1):
    final = 1
    print(f'{n}! = ',end='')
    for c in range(n,0,-1):
        if c > 1:
            print(f'{c} X ',end='')
        if c == 1:
            print(f'{c} ',end='')
        final *= c
    print(f'= {final}')


fatorial(n = int(input('Numero: ')))
```

Guanabara)

Na solução do professor ele mostra que podemos chamar a função várias vezes se quisermos vários resultados diretos ou por input, porém ele não mostra o print da descrição, apenas do resultado.

Usando o método return atribuimos um valor calculado localmente numa def a uma variável global.

```
def fatorial(num=1):
    f = 1
    for c in range (num, 0 , -1):
        f *= c
    return f

f1 = fatorial(5)
f2 = fatorial(4)
f3 = fatorial()
print(f'Os resultados são {f1}, {f2} e {f3}')
```

2) Verificados se o número é par ou ímpar

a)

Existem algumas formas de fazer esta solução mas vou fazer da forma que o professor realizou que foi criar uma def que o retorno dela seja a resposta em booleano, se sim ou se não, sendo assim:

```
def PAR(n=0):
    if n % 2 == 0:
        return True
    else:
        return False

num = int(input('Digite um número: '))
print(PAR(num))
```

Podemos aprimorar para informar na tela se ele é par ou impar ao invés de true ou false.

Desta forma ele dá um print, mas e se quiséssemos que seja verificar o return do valor booleano ?

```
def PAR(n=0):
    if n % 2 == 0:
        print(f'{n} é PAR')
    else:
        print(f'{n} é IMPAR')

num = int(input('Digite um número: '))
PAR(num)
```

Poderiamos fazer desta forma:

```
def par(n=0):
    if n % 2 == 0:
        return True
    else:
        return False

n = int(input('Digite um número: '))
res = par(n)
if res == True:
    print(f'{n} = PAR')
else:
    print(f'{n} = IMPAR')
```

Podemos testar o valor booleano diretamente na chamada da função, que o código testa sempre o verdadeiro primeiro.

```
def par(n=0):
    if n % 2 == 0:
        return True
    else:
        return False
```

```
n = int(input('Digite um número: '))
if par(n):
    print(f'{n} = PAR')
else:
    print(f'{n} = IMPAR')
```

3) Crie um programa que tenha uma função chamada voto() que vai receber como parâmetro o ano de nascimento de uma pessoa, retornando um valor literal indicando se uma pessoa tem voto negado, opcional ou obrigatório nas eleições

- < 18 é negado
- >= 18 é obrigatório
- 65 é opcional

a)

Neste caso adicionamos a idade diretamente, mas o intuito é o programa calcular a idade

```
def voto(n=0):
    if n < 18:
        return 'Voto negado'
    elif n >= 18:
        return 'Voto obrigatório'
    elif n > 65:
        return 'Voto opcional'

idade = int(input('Idade: '))
resultado = voto(idade)
print(resultado)
```

b)

Importei a biblioteca datetime para ele puxar a data e hora do computador e poder verificar o ano, após isso fiz o calculo da diferença do ano atual para o digitado.

```
from datetime import date

def voto(n=0):
    hoje = date.today().year
    idade = hoje - n
    print(f'Atualmente você tem {idade} anos e seu voto é ',end='')
    if idade < 18:
        return 'Negado'
    else:
        if idade > 65:
            return 'Opcional'
```

```
        if idade > 18:
            return 'Obrigatório'

resultado = voto(n = int(input('Ano nascimento: ')))
print(resultado)
```

c)

Podemos trabalhar com cores também da seguinte forma:

```
from datetime import date
from time import sleep

def analise():
    print('\n{}{:^100}'.format('\033[30;43m','Verificando dados'))
    for c in range(4):
        print('{:^100}'.format('.'))
        sleep(0.5)
    print('\033[m')

def voto(n=0):
    analise()
    hoje = date.today().year
    idade = hoje - n
    print('\033[30;46m',end='')
    print('{:^100}'.format('RESULTADO'))
    print('\033[m',end='')
    print()
    print(f'Atualmente você tem {idade} anos e seu voto é ',end='')
    if idade < 18:
        return 'Negado'
    else:
        if idade > 65:
            return 'Opcional'
        if idade > 18:
            return 'Obrigatório'

print('{}{:^100}{}'.format('\033[7;40m','VERIFICADOR DE VOTO','\033[m'))
resultado = voto(n = int(input('{}{}{} '.format('\033[4m','Ano nasciment
o:','\033[m'))))
print(resultado)
```

Guanabara)

Neste exemplo, o Guanabara utilizou a biblioteca de datetime apenas dentro da função, sem colocar no global do código, economizando memória mesmo que apenas um pouco, fora isso, ele determinou que:

- < 16 não pode votar
- >= 16 e < 18 and > 65 é opcional
- restante é obrigatório

```
def voto(ano):
    from datetime import date
    atual = date.today().year
    idade = atual - ano
    if idade < 16:
        return f'com {idade} anos: NÃO VOTA.'
    elif 16 <= idade < 18 or idade > 65:
        return f'com {idade} anos: VOTO OPCIONAL.'
    else:
        return f'com {idade} anos: VOTO OBRIGATÓRIO.'


print(voto(2000))
```

4) Crie um programa que tenha uma função fatorial() que receba dois parâmetros: o primeiro que indique o número a calcular e o outro chamado show que será um valor lógico (opcional) indicando se será mostrado ou não na tela o processo de calculo do fatorial

a)

```
def fatorial(n=1,show = False):
    """
    -> Calcula o Fatorial de um número
    :param n: O número a ser calculado
    :param show: (opcional, padrão False) Mostrar ou não a conta
    :return: O valor do Fatorial de um número n
    """
    print('-'*30)
    fim = 1
    if show:
        print(f'{n}! = ',end='')
        for c in range(n,0,-1):
            if c > 1:
                print(f'{c} x ',end='')
            else:
                print(f'{c} ',end='')
            fim *= c
        print(f'= {fim}')
    else:
        for c in range(n,0,-1):
```

```
            fim *= c
        print(f'{n}! = {fim}')


fatorial(10,False)
```

Guanabara)

A forma que o professor fez foi um pouco diferente, ele inseriu o show dentro do for poupando tamanho do código, então está funcionando assim:

- ele printa o número indicando que é o fatorial (5 ! =)
- f recebe 1
- for c in range de inicio n, fim de 0 e passo -1
- se o show for verdadeiro então ele printa o c, se o c for maior que 1 ele printa x senão ele printa =
- f = f * c
- retorna o valor de f
- Mostra o calculo ou não dependendo do show

Achei essa forma legal, pois ao invés de condicionar e mostrar o número com o caracter, basta fazer a verificação do número iterante e mostrar o caracter em si dependendo do seu valor, assim dá pra controlar melhor o que é mostrado na tela ou não.

```
def fatorial(n, show=False):
    """
    -> Calcula o Fatorial de um número
    :param n: O número a ser calculado
    :param show: (opcional, padrão False) Mostrar ou não a conta
    :return: O valor do Fatorial de um número n
    """
    print(f'{n}! = ',end='')
    f = 1
    for c in range(n,0,-1):
        if show:
            print(c,end='')
            if c > 1:
                print(f' x ',end='')
            else:
                print(' = ',end='')
        f *= c
    return f

print(fatorial(5,True))
```

5) Faça um programa que tenha uma função chamada ficha() que receba dois parâmetros opcionais: o nome de um jogador e quantos gols ele marcou. O programa deverá ser capaz de mostrar a ficha do jogador, mesmo que algum dado não tenha sido informado corretamente

a)

Esta abordagem faz o seguinte, eu tenho a função ficha que receberá o nome e a quantidade de gols digitada pelo usuário, porém para que o programa aceite o valor vazio e mostre os resultados digitados, eu preciso realizar a verificação em algum momento, neste exemplo eu fiz após o input e no final chamei a função, mas existe outra forma de resolver.

```
def ficha(nome,gols):
    print('-'*30)
    print(f'O jogador {nome} fez {gols} gol(s) no campionato')

jogador = input('Nome do jogador: ').strip()
n_gols = input('Números de Gols: ').strip()

if jogador == '':
    jogador = '<desconhecido>'
if n_gols == '':
    n_gols = 0
else:
    n_gols = int(n_gols)

ficha(jogador,n_gols)
```

b)

Esta outra forma faz a verificação por dentro da função, então eu não preciso por exemplo gastar várias linhas de código na main e deixar mais organizado.

```
def ficha(nome,gols):
    if nome == '':
        nome = '<desconhecido>'
    if gols == '':
        gols = 0
    else:
        gols = int(n_gols)
    print('-'*30)
    print(f'O jogador {nome} fez {gols} gol(s) no campionato')

jogador = input('Nome do jogador: ').strip()
n_gols = input('Números de Gols: ').strip()
ficha(jogador,n_gols)
```

Guanabara)

A forma dele resolver também foi diferente, vamos ver:

- Começando pela parte global ele deu dois inputs nas variáveis n e g considerando elas como string
- Abaixo fez a verificação da seguinte forma:
  - Se g for numérica então, será convertida para int, senão g valerá 0
  - Se n.strip for nada, então ele passa para a função ficha o gol local valendo o g
  - Senão ele passa para a função ficha o n e g
- Dentro da função declaramos que jog é opcional e valerá '<desconhecido>' se não for passado e gol valerá 0 como padrão
- No final printamos as informações.

Parte interessante desta resolução foi a verificação, ele escolher parâmetro que passar ou não, então no global podemos determinar qual parâmetro será passado para o local desde que o outro tenha um valor opcional se não for preenchido.

```
def ficha(jog = '<desconhecido>',gol=0):
    print(f'O jogador {jog} fez {gol} gol(s) no campeonato')


n = input('Nome de Jogador: ')
g = input('Número de Gols: ')
if g.isnumeric():
    g = int(g)
else:
    g = 0
if n.strip() == '':
    ficha(gol=g)
else:
    ficha(n,g)
```

6) Crie um programa que tenha a função leiaInt() que vai funcionar de forma semelhante à função input() do python, só que fazendo a validação para aceitar apenas um valor numérico. Ex n = leiaInt('Digite um n')

a)

```
def leiaInt(n):
    while n.isnumeric() == False:
        n = input('{}{}{}'.format('\033[31m','Erro! Favor digitar apenas va
lores numéricos: ','\033[m'))
    print(f'O valor {n} é numérico')
```

```
res = leiaInt(n = input('Digite um número: '))
```

Guanabara)

Cara, o professor fez algo muito doido, completamente diferente e sem necessidade, mas entendi que é para mostrar de quais formas podemos continuar manipulando e jogando valores para dentro de outras coisas.

Então para entender:

- Na parte global a variável n recebe a chamada de LeiaInt passando como parâmetro 'Digite um número: '
- Dentro da def LeitaInt ele recebe o conteúdo e armazena em msg
  - É declarada a variável ok com falso, ela vai ser a controladora para passar para a próxima etapa
  - variável varol recebe 0
  - Agora entramos num loop infinito onde:
    - n recebe o input e é mostrado a msg
    - se n for numérico, então valor recebe uma cópia de n e converte para inteiro
    - ok recebe true
    - senão ele printa a mensagem de erro
    - se ok for true então ele sai do loop
  - retorna o valor
- No final ele mostra o conteúdo

```
def leiaInt(msg):
    ok = False
    valor = 0
    while True:
        n = input(msg)
        if n.isnumeric():
            valor = int(n)
            ok = True
        else:
            print('\033[0;31mErro! Digite um número inteiro válido.\033[m')
        if ok:
            break
    return valor

n = leiaInt('Digite um número: ')
print(f'Você acabou de digitar o número {n}')
```

7) Faça um programa que tenha uma função notas() que pode receber várias notas de alunos e vai retornando um dicionário com as seguinte informações:

- Quantidade de notas
- a maior nota
- a menor nota
- a méd ia da turma
- a situação (opcional)

Adicione também as docstrings da função

a)

Ambos foram bem simples, mas no meu caso não teria necessidade de eu criar outra lista, porque o *num já cria uma tupla, então era só transportar esses valores para o dicionário, mas independente funcionou.

```
def notas(*num,sit=False):
    notas = list()
    info = dict()
    for i, c in enumerate(num):
        notas.append(num[i])
    info['Total'] = len(notas)
    info['Maior'] = max(notas)
    info['Menor'] = min(notas)
    info['Media'] = sum(notas)/len(notas)
    if sit:
        if 6 < info['Media'] < 7:
            info['Situação'] = 'Razoável'
        if info['Media'] < 6:
            info['Situação'] = 'Ruim'
        if info['Media'] > 7:
            info['Situação'] = 'Boa'
    return info

resp = notas(6,10,6,sit=True)
print(resp)
```

Guanabara)

```
def notas(*n,sit=False):
    """
    -> Função para analisar notas e situações de vários alunos.
    :param n: uma ou mais notas dos aluos (aceita várias).
    :param sit: valor opcional, indicando se deve ou não adicionar a situaç
ão.
```

```
    :return: dicionário com várias informações sobre a situação da turma.
    """
    r = dict()
    r['total'] = len(n)
    r['Maior'] = max(n)
    r['Menor'] = min(n)
    r['Media'] = sum(n)/len(n)
    if sit:
        if r['Media'] > 7:
            r['Situação'] = 'Boa'
        elif r['Media'] >= 5:
            r['Situação'] = 'Razoável'
        else:
            r['Situação'] = 'Ruim'
    return(r)



# Programa Principal
resp = notas(5.5, 2.5, 9, 8.5)
print(resp)
help(notas)
```

8) Faça um mini sistema que utilize o interactive help do python. O usuário vai digitar o comando e o manual vai aparecer. Quando o usuário digitar a palavra “fim” o programa se encerrerá. Use cores.

a)

Podemos resolver com dicionário então chamamos a cor pelo nome

```
from time import sleep

# Definição de cores para utilização no terminal

cor = {
    'sem cor': '\033[m',
    'vermelho': '\033[0;30;41m',
    'verde': '\033[0;30;42m',
    'amarelo': '\033[0;30;43m',
    'azul': '\033[0;30;44m',
    'roxo': '\033[0;30;45m',
    'branco': '\033[7;40m'
}


# Programa para printar conteúdo editado e com cor
```

```
def mensagem(msg,c='sem cor'):
    print(cor[c],end='')
    tamanho = len(msg)+4
    print('-'*tamanho)
    print(f'  {msg}')
    print('-'*tamanho)
    print(cor['sem cor'],end='')
    sleep(1)


# Programa para pesquisar e mostrar os comandos do python

def pesquisa(com):
    mensagem(f'Acessando o manual do comando \'{com}\'','amarelo')
    print(cor['branco'],end='')
    help(com)
    print(cor['sem cor'],end='')
    sleep(1)


# Programa principal

comando = ''
while True:
    mensagem('SISTEMA DE AJUDA PyHELP','verde')
    comando = input('Fundção ou Biblioteca >')
    if comando.upper()=='FIM':
        break
    else:
        pesquisa(comando)
mensagem('ATÉ LOGO','vermelho')
```

Guanabara)

Vamos entender este programa, pois ficou um pouco complexo e preciso aprender antes que siga para a próxima etapa de modulação.

Conceito do script:

- Ele quer algo colorido onde o programa identifique o que escrevemos use o input para o comando help do python, trazendo toda a documentação da função ou da biblioteca

Entendendo o código:

1. feito a importação do sleep vinda da biblioteca time
2. declaramos no início do código uma tupla (não sofrerá alterações), poderia ser também uma lista ou um dicionário (vou tentar reproduzir, colocando o nome em cada cor), ele salvou as básicas sem alterar a fonte, apenas o fundo.

Função TITULO( )

3. Iniciamos o aprendizado com a função TÍTULO , ela é o que mostrar as mensagens com a cor especificada, vamos entender sua estrutura:
    a. def título(msg,cor=0) → a função recebe dois parâmetros, um msg sendo obrigatório e a cor é opcional, sendo ela por padrão 0, caso não seja preenchida
    b. primeiro print ele delimita o inicio da cor escolhida em c[cor] então se tiver entre 1 e 6 ele já vai colorir, porém ainda não colocando um fim
    c. variável tamanho recebe o tamanho da mensgem + 4, ou seja ele vai se adaptar dependendo do tamanho da mensagem digitada
    d. print('~'*tamanho') → ele insere o simbolo conforme tamanho anteriormente definido
    e. print(f' {msg}') → mostra a mensagem com 2 espaços de forma à esquerda (poderiamos centralizar se quiséssemos)
    f. print('~'*tamanho') → ele insere o simbolo conforme tamanho anteriormente definido
    g. print(c[0],end='') → delimita o final da cor da mensagem, não formatando o código inteiro
    h. sleep(1) → faz uma pausa de 1 segundo

Função AJUDA( )

4. A função ajuda recebe um parâmetro obrigatório sendo o comando que o usuário solicitar a documentação, vamos entender o passo a passo dele então:
    a. def ajuda(com) → recebe o comando escolhido pelo usuário (ou digitado de forma errada)
    b. chama a função título passando uma mensagem como a principal e determinando a cor sendo a 4 que neste caso é a azul
    c. print(c[6],end='') → determina o início do bloco de cor sendo branco
    d. help(com) → chama a função interna do python com toda a documentação da dunção ou biblioteca pesquisada
    e. print(c[0], endo='') → determina o final da cor sendo o fina do documento apresentado
    f. sleep(1) → dá um sleep de 1 segundo

Main / Código princial

5. declaração da variável comando iniciando zerada
6. while true → inicio de um loop infinito, então em algum momento teremos um break para parar o código e dentro dele, temos o seguinte:
    a. título('SISTEMA DE AJUDA PyHELP',2) → chamada da função título com a cor 2 como parâmetro
    b. comando recebe o input de função ou biblioteca
    c. se comando em maiúsculo for 'FIM' então ele dá um break
    d. senão, ajuda(comando)

Finaliza com o titulo até logo.

```
from time import sleep

c = ('\033[m',        # 0 - sem cores
     '\033[0;30;41m', # 1 - vermelho
     '\033[0;30;42m', # 2 - verde
     '\033[0;30;43m', # 3 - amarelo
     '\033[0;30;44m', # 4 - azul
     '\033[0;30;45m', # 5 - roxo
     '\033[7;40m'     # 6 - branco
);

def título(msg,cor=0):
    print(c[cor],end='')
    tamanho = len(msg)+4
    print('~'*tamanho)
    print(f'  {msg}')
    print('~'*tamanho)
    print(c[0],end='')
    sleep(1)

def ajuda(com):
    título(f'Acessando o manual do comando \'{com}\'',4)
    print(c[6],end='')
    help(com)
    print(c[0],end='')
    sleep(1)


# Programa principal
comando = ''
while True:
    título('SISTEMA DE AJUDA PyHELP',2)
    comando = input('Função ou Biblioteca > ')
    if comando.upper() == 'FIM':
        break
    else:
        ajuda(comando)
título('ATÉ LOGO!',1)
```

b)

Esta foi a forma como eu fiz, com listas e chamando uma main para rodar o código

```
from time import sleep

c = [
    '\033[m',        # 0 - sem formatação
```

```
    '\033[0;30;41m', # 1 - vermelho
    '\033[0;30;42m', # 2 - verde
    '\033[0;30;43m', # 3 - amarelo
    '\033[0;30;44m', # 4 - azul
    '\033[0;30;45m',  # 5 - roxo
    '\033[0;30;46m',  # 6 - azul bebe
    '\033[0;30;47m',  # 7 - cinza
    '\033[7;40m'      # 8 - branco
]

def mensagem(texto,cor=0):
    """
    -> Função para mostrar no terminar uma mensagem de uma cor específica
    :param texto: mensagem digitada pelo usuário ou determinada pelo progra
ma
    :param cor: uma das cores da lista cor
    :return: sem return, apenas printa a mensagem
    """
    print(c[cor],end='')
    tamanho = len(texto)+5
    print('='*tamanho)
    print('{:^{}}'.format(texto,tamanho))
    print('='*tamanho)
    print(c[0],end='')

def menu_ajuda(com):
    """
    -> Função serve para mostrar o comando pesquisado no help
    :param com: comando digitado pelo usuário
    :return: sem return, apenas printa o comando
    """
    mensagem('Analisando o comando digitado',3)
    print(c[3],end='')
    for item in range(4):
        print('{:^29}'.format('.'))
        sleep(0.5)
    print(c[0],end='')
    print(c[8],end='')
    help(com)
    print(c[0],end='')
    sleep(1)

def main():
    """
    -> Serve para a execução principal do programa, aqui utilizamos todas a
s funções em um só lugar.
    :return: sem return, apenas mostra na tela.
    """
```

```
    comando = ''
    while True:
        mensagem('PROGRAMA AJUDA COM O PYTHON',2)
        comando = input('Digite uma função ou biblioteca > ')
        if comando.upper() == 'FIM':
            break
        else:
            menu_ajuda(comando)
    mensagem('FIM DE EXECUÇÃO !',7)

main()
```