Exercicios_Fixacao2

0.1 Unidade 2 - Introdução ao Phyton

0.2 Exercícios de Fixação

Aluno: Cleverson Guandalin

0.2.1 Exercícios sobre Estruturas de Controle de Decisão

Disponiveis em: https://wiki.python.org.br/EstruturaDeDecisao

16 - Faça um programa que calcule as raízes de uma equação do segundo grau, na forma ax2 + bx + c. O programa deverá pedir os valores de a, b e c e fazer as consistências, informando ao usuário nas seguintes situações:

```
a) Se o usuário informar o valor de A igual a zero, a equação não é do segundo grau e o progra
b) Se o delta calculado for negativo, a equação não possui raizes reais. Informe ao usuário e
c) Se o delta calculado for igual a zero a equação possui apenas uma raiz real; informe-a ao us
d) Se o delta for positivo, a equação possui duas raiz reais; informe-as ao usuário;
```

[3]:
```
# pede os valores de a, b e c ao usuário
a = float(input("Digite o valor de a: "))
b = float(input("Digite o valor de b: "))
c = float(input("Digite o valor de c: "))

# verifica se a é diferente de zero, pois se for, não é uma equação do segundo␣
 ↪grau
if a == 0:
    print("Não é uma equação do segundo grau!")
else:
    # calcula o delta da equação
    delta = b**2 - 4*a*c

    # verifica se o delta é negativo, pois neste caso a equação não possui␣
 ↪raízes reais
    if delta < 0:
        print("A equação não possui raízes reais!")
    # verifica se o delta é igual a zero, pois neste caso a equação possui␣
 ↪apenas uma raiz real
    elif delta == 0:
        x = (-b + (delta ** 0.5)) / (2*a)
```

```
        print("A equação possui apenas uma raiz real: ", x)
    # se o delta for positivo, a equação possui duas raízes reais
    else:
        x1 = (-b + (delta ** 0.5)) / (2*a)
        x2 = (-b - (delta ** 0.5)) / (2*a)
        print("A equação possui duas raízes reais: ", x1, " e ", x2)
```

```
Digite o valor de a: 10
Digite o valor de b: 12
Digite o valor de c: 2
A equação possui duas raízes reais:  -0.2  e  -1.0
```

17 - Faça um Programa que peça um número correspondente a um determinado ano e em seguida informe se este ano é ou não bissexto.

[5]:
```
# pede ao usuário para digitar um ano
ano = int(input("Digite um ano: "))

# verifica se o ano é bissexto
if (ano % 4 == 0 and ano % 100 != 0) or ano % 400 == 0:
    print(ano, " é um ano bissexto!")
else:
    print(ano, " não é um ano bissexto!")
```

```
Digite um ano: 2024
2024  é um ano bissexto!
```

18 - Faça um Programa que peça uma data no formato dd/mm/aaaa e determine se a mesma é uma data válida.

[6]:
```
# pede ao usuário para digitar uma data no formato dd/mm/aaaa
data = input("Digite uma data no formato dd/mm/aaaa: ")

# verifica se a data tem o formato correto (dd/mm/aaaa)
if len(data) != 10 or data[2] != "/" or data[5] != "/":
    print("Data inválida!")
else:
    # extrai o dia, mês e ano da data
    dia = int(data[:2])
    mes = int(data[3:5])
    ano = int(data[6:])

    # verifica se o dia e o mês são válidos
    if dia < 1 or mes < 1 or mes > 12:
        print("Data inválida!")
    elif mes == 2:
        if dia > 29 or (dia == 29 and not (ano % 4 == 0 and ano % 100 != 0) or␣
 ↪ano % 400 == 0):
            print("Data inválida!")
```

```
        else:
            print("Data válida!")
    elif mes in [4, 6, 9, 11]:
        if dia > 30:
            print("Data inválida!")
        else:
            print("Data válida!")
    else:
        if dia > 31:
            print("Data inválida!")
        else:
            print("Data válida!")
```

```
Digite uma data no formato dd/mm/aaaa: 30/06/1985
Data válida!
```

19 - Faça um Programa que leia um número inteiro menor que 1000 e imprima a quantidade de centenas, dezenas e unidades do mesmo.

```
Observando os termos no plural a colocação do "e", da vírgula entre outros. Exemplo:
326 = 3 centenas, 2 dezenas e 6 unidades
12 = 1 dezena e 2 unidades Testar com: 326, 300, 100, 320, 310,305, 301, 101, 311, 111, 25, 20
```

[7]:
```
numero = int(input("Digite um número inteiro menor que 1000: "))

if numero >= 1000:
    print("Número inválido!")
else:
    # Separação das centenas, dezenas e unidades utilizando operações␣
 ↪matemáticas
    centenas = numero // 100
    dezenas = (numero % 100) // 10
    unidades = (numero % 100) % 10

    # Verificações para imprimir corretamente as palavras no plural ou singular
    if centenas == 1:
        print(f"{centenas} centena", end="")
    elif centenas > 1:
        print(f"{centenas} centenas", end="")

    if centenas > 0 and (dezenas > 0 or unidades > 0):
        print(" e ", end="")

    if dezenas == 1:
        print(f"{dezenas} dezena", end="")
    elif dezenas > 1:
        print(f"{dezenas} dezenas", end="")
```

```
    if dezenas > 0 and unidades > 0:
        print(" e ", end="")

    if unidades == 1:
        print(f"{unidades} unidade", end="")
    elif unidades > 1:
        print(f"{unidades} unidades", end="")
```

```
Digite um número inteiro menor que 1000: 850
8 centenas e 5 dezenas
```

20 - Faça um Programa para leitura de três notas parciais de um aluno. O programa deve calcular a média alcançada por aluno e presentar:

```
a) A mensagem "Aprovado", se a média for maior ou igual a 7, com a respectiva média alcançada;
b) A mensagem "Reprovado", se a média for menor do que 7, com a respectiva média alcançada;
c) A mensagem "Aprovado com Distinção", se a média for igual a 10.
```

[8]:
```
# Leitura das notas
nota1 = float(input("Digite a primeira nota: "))
nota2 = float(input("Digite a segunda nota: "))
nota3 = float(input("Digite a terceira nota: "))

# Cálculo da média
media = (nota1 + nota2 + nota3) / 3

# Verificação das condições de aprovação
if media == 10:
    print(f"Aprovado com Distinção. Média: {media:.2f}")
elif media >= 7:
    print(f"Aprovado. Média: {media:.2f}")
else:
    print(f"Reprovado. Média: {media:.2f}")
```

```
Digite a primeira nota: 7
Digite a segunda nota: 8
Digite a terceira nota: 10
Aprovado. Média: 8.33
```

0.2.2 Exercícios sobre Estruturas de Repetição

Disponiveis em: https://wiki.python.org.br/EstruturaDeRepeticao

10 - Faça um programa que receba dois números inteiros e gere os números inteiros que estão no intervalo compreendido por eles.

[10]:
```
# recebendo os números do usuário
num1 = int(input("Digite o primeiro número: "))
num2 = int(input("Digite o segundo número: "))
```

```
# verificando qual é o menor e o maior número
if num1 < num2:
    menor = num1
    maior = num2
else:
    menor = num2
    maior = num1

# percorrendo o intervalo entre os números e imprimindo-os
for i in range(menor+1, maior):
    print(i, end=' ')
```

```
Digite o primeiro número: 20
Digite o segundo número: 50
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49
```

11 - Altere o programa anterior para mostrar no final a soma dos números.

[11]:
```
# recebendo os números do usuário
num1 = int(input("Digite o primeiro número: "))
num2 = int(input("Digite o segundo número: "))

# verificando qual é o menor e o maior número
if num1 < num2:
    menor = num1
    maior = num2
else:
    menor = num2
    maior = num1

soma = 0  # variável para armazenar a soma dos números

# percorrendo o intervalo entre os números e imprimindo-os
for i in range(menor+1, maior):
    print(i, end=' ')
    soma += i

print("\nA soma dos números é:", soma)
```

```
Digite o primeiro número: 10
Digite o segundo número: 20
11 12 13 14 15 16 17 18 19
A soma dos números é: 135
```

12 - Desenvolva um gerador de tabuada, capaz de gerar a tabuada de qualquer número inteiro entre 1 a 10. O usuário deve informar de qual numero ele deseja ver a tabuada. A saída deve ser conforme o exemplo abaixo:

```
Tabuada de 5:
```

```
5 X 1 = 5
5 X 2 = 10
...
5 X 10 = 50
```

[12]:
```python
# Recebendo o número da tabuada do usuário
num = int(input("Digite um número para ver a sua tabuada: "))

print("Tabuada de", num, ":")

# Percorrendo os números de 1 a 10 e imprimindo a tabuada
for i in range(1, 11):
    print(num, "X", i, "=", num*i)
```

```
Digite um número para ver a sua tabuada: 6
Tabuada de 6 :
6 X 1 = 6
6 X 2 = 12
6 X 3 = 18
6 X 4 = 24
6 X 5 = 30
6 X 6 = 36
6 X 7 = 42
6 X 8 = 48
6 X 9 = 54
6 X 10 = 60
```

13 - Faça um programa que peça dois números, base e expoente, calcule e mostre o primeiro número elevado ao segundo número. Não utilize a função de potência da linguagem.

[13]:
```python
# Pede ao usuário para informar os números
base = int(input("Digite a base: "))
expoente = int(input("Digite o expoente: "))

# Inicializa o resultado como 1
resultado = 1

# Calcula a potência usando um laço de repetição
for i in range(expoente):
    resultado *= base

# Exibe o resultado
print(f"{base} elevado a {expoente} é igual a {resultado}")
```

```
Digite a base: 2
Digite o expoente: 4
2 elevado a 4 é igual a 16
```

14 - Faça um programa que peça 10 números inteiros, calcule e mostre a quantidade de números pares e a quantidade de números impares.

[14]:
```
# Inicializa as variáveis que contarão a quantidade de números pares e ímpares
qtd_pares = 0
qtd_impares = 0

# Pede ao usuário para informar os números
for i in range(10):
    num = int(input(f"Digite o {i+1}º número: "))

    # verifica se o número é par ou ímpar e incrementa a variável correspondente
    if num % 2 == 0:
        qtd_pares += 1
    else:
        qtd_impares += 1

# Exibe o resultado
print(f"Foram informados {qtd_pares} números pares e {qtd_impares} números␣
 ↪ímpares.")
```

```
Digite o 1º número: 2
Digite o 2º número: 4
Digite o 3º número: 5
Digite o 4º número: 7
Digite o 5º número: 8
Digite o 6º número: 9
Digite o 7º número: 10
Digite o 8º número: 11
Digite o 9º número: 15
Digite o 10º número: 12
Foram informados 5 números pares e 5 números ímpares.
```

15 - A série de Fibonacci é formada pela seqüência 1,1,2,3,5,8,13,21,34,55,... Faça um programa capaz de gerar a série até o n−ésimo termo.

[17]:
```
# Pede ao usuário para informar o número de termos da sequência de Fibonacci
n = int(input("Digite o número de termos da sequência de Fibonacci: "))

# Inicializa os dois primeiros termos da sequência
fibonacci_anterior = 1
fibonacci_atual = 1

# Exibe os dois primeiros termos da sequência
print(fibonacci_anterior)
print(fibonacci_atual)

# Calcula os demais termos da sequência usando um laço de repetição
for i in range(2, n):
    fibonacci_proximo = fibonacci_anterior + fibonacci_atual
    fibonacci_anterior = fibonacci_atual
```

```
    fibonacci_atual = fibonacci_proximo
    print(fibonacci_atual)
```

```
Digite o número de termos da sequência de Fibonacci: 20
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

0.2.3 Exercícios sobre Funções

Disponiveis em: https://wiki.python.org.br/ExerciciosFuncoes

10 - Jogo de Craps. Faça um programa de implemente um jogo de Craps. O jogador lança um par de dados, obtendo um valor entre 2 e 12. Se, na primeira jogada, você tirar 7 ou 11, você um "natural" e ganhou. Se você tirar 2, 3 ou 12 na primeira jogada, isto é chamado de "craps" e você perdeu. Se, na primeira jogada, você fez um 4, 5, 6, 8, 9 ou 10,este é seu "Ponto". Seu objetivo agora é continuar jogando os dados até tirar este número novamente. Você perde, no entanto, se tirar um 7 antes de tirar este Ponto novamente.

[18]:
```
import random

def craps():
    # Jogada inicial
    jogada = sum(random.randint(1, 6) for _ in range(2))

    if jogada in (7, 11):
        print(f"Você tirou {jogada}, um natural! Você ganhou!")
    elif jogada in (2, 3, 12):
        print(f"Você tirou {jogada}, craps! Você perdeu.")
    else:
```

```
        print(f"Seu ponto é {jogada}. Jogue os dados novamente até tirar␣
↪{jogada} ou 7.")

        # continuar jogando até ganhar ou perder
        while True:
            nova_jogada = sum(random.randint(1, 6) for _ in range(2))
            print(f"Você tirou {nova_jogada}.")

            if nova_jogada == jogada:
                print(f"Você tirou {jogada} novamente! Você ganhou!")
                break
            elif nova_jogada == 7:
                print("Você tirou 7 antes de tirar seu ponto novamente. Você␣
↪perdeu.")
                break

craps()
```

```
Seu ponto é 6. Jogue os dados novamente até tirar 6 ou 7.
Você tirou 5.
Você tirou 2.
Você tirou 6.
Você tirou 6 novamente! Você ganhou!
```

11 - Data com mês por extenso. Construa uma função que receba uma data no formato DD/MM/AAAA e devolva uma string no formato D de mesPorExtenso de AAAA. Opcionalmente, valide a data e retorne NULL caso a data seja inválida.

[19]:
```
from datetime import datetime

def data_por_extenso(data):
    try:
        data_objeto = datetime.strptime(data, '%d/%m/%Y')
        mes_extenso = data_objeto.strftime('%B') # obtém o mês por extenso
        return f"{data_objeto.day} de {mes_extenso} de {data_objeto.year}"
    except ValueError:
        return None # retorna None se a data é inválida

print(data_por_extenso('31/12/2022')) # imprime "31 de December de 2022"
print(data_por_extenso('32/12/2022')) # imprime "None"
```

```
31 de December de 2022
None
```

12 - Embaralha palavra. Construa uma função que receba uma string como parâmetro e devolva outra string com os carateres embaralhados. Por exemplo: se função receber a palavra python, pode retornar npthyo, ophtyn ou qualquer outra combinação possível, de forma aleatória. Padronize em sua função que todos os caracteres serão devolvidos em caixa alta ou caixa baixa, independentemente de como foram digitados.

[21]:
```python
import random

def embaralha_palavra(palavra):
    """Embaralha os caracteres de uma palavra e retorna uma nova string."""
    # Converte a palavra para caixa baixa
    palavra = palavra.lower()
    # Converte a palavra para uma lista de caracteres
    caracteres = list(palavra)
    # Embaralha a lista de caracteres
    random.shuffle(caracteres)
    # Retorna a lista de caracteres como uma string
    return ''.join(caracteres)

# Exemplo de uso
palavra = 'Python'
palavra_embaralhada = embaralha_palavra(palavra)
print(palavra_embaralhada)
```

yhpnot

13 - Desenha moldura. Construa uma função que desenhe um retângulo usando os caracteres '+' , '—' e '| '. Esta função deve receber dois parâmetros, linhas e colunas, sendo que o valor por omissão é o valor mínimo igual a 1 e o valor máximo é 20. Se valores fora da faixa forem informados, eles devem ser modificados para valores dentro da faixa de forma elegante.

[20]:
```python
def desenha_moldura(linhas=1, colunas=1):
    """Desenha um retângulo com os caracteres +, - e |."""
    # Verifica se os valores de linhas e colunas estão dentro da faixa permitida
    if linhas < 1 or linhas > 20:
        linhas = 1
    if colunas < 1 or colunas > 20:
        colunas = 1

    # Desenha a moldura
    for i in range(linhas):
        for j in range(colunas):
            if i == 0 or i == linhas-1:
                # Primeira e última linhas
                if j == 0 or j == colunas-1:
                    # Canto superior esquerdo, canto superior direito,
                    # canto inferior esquerdo e canto inferior direito
                    print('+', end='')
                else:
                    # Demais posições da primeira e última linhas
                    print('-', end='')
            else:
                # Demais linhas
                if j == 0 or j == colunas-1:
```

```
                    # Primeira e última colunas
                    print('|', end='')
                else:
                    # Demais posições
                    print(' ', end='')
        print()  # Pula para a próxima linha

# Exemplo
desenha_moldura(5, 8)
```

```
+------+
|      |
|      |
|      |
+------+
```

14 - Quadrado mágico. Um quadrado mágico é aquele dividido em linhas e colunas, com um número em cada posição e no qual a soma das linhas, colunas e diagonais é a mesma. Por exemplo, veja um quadrado mágico de lado 3, com números de 1 a 9:

```
8  3  4
1  5  9
6  7  2
```

Elabore uma função que identifica e mostra na tela todos os quadrados mágicos com as caracterís

[3]:
```
# Definir uma função que verifica se uma lista de 9 números forma um quadrado␣
 ↪mágico de lado 3
def eh_quadrado_magico(lista):
    # Verificar se a soma das linhas é igual a 15
    if sum(lista[0:3]) != 15 or sum(lista[3:6]) != 15 or sum(lista[6:9]) != 15:
        return False
    # Verificar se a soma das colunas é igual a 15
    if sum(lista[0::3]) != 15 or sum(lista[1::3]) != 15 or sum(lista[2::3]) !=␣
 ↪15:
        return False
    # Verificar se a soma das diagonais é igual a 15
    if sum(lista[0::4]) != 15 or sum(lista[2:7:2]) != 15:
        return False
    # Se todas as condições forem satisfeitas, retornar True
    return True

# Definir uma função que gera todas as permutações possíveis de uma lista␣
 ↪usando recursão
def gerar_permutacoes(lista):
    # Se a lista tiver apenas um elemento, retornar uma lista com esse elemento
    if len(lista) == 1:
        return [lista]
```

```
    # Senão, para cada elemento da lista, remover esse elemento e gerar as␣
 ↪permutações do resto da lista
    else:
        permutacoes = []
        for i in range(len(lista)):
            elemento = lista[i]
            resto = lista[:i] + lista[i+1:]
            # Para cada permutação do resto da lista, adicionar o elemento no␣
 ↪início e adicionar à lista de permutações
            for p in gerar_permutacoes(resto):
                permutacoes.append([elemento] + p)
        # Retornar a lista de permutações
        return permutacoes

# Gerar todas as permutações possíveis de 1 a 9 usando a função␣
 ↪gerar_permutacoes
permutacoes = gerar_permutacoes(list(range(1,10)))

# Inicializar uma variável para contar o número de quadrados mágicos encontrados
contador = 0

# Para cada permutação, verificar se ela forma um quadrado mágico e mostrar na␣
 ↪tela se for o caso
for p in permutacoes:
    if eh_quadrado_magico(p):
        # Incrementar o contador e mostrar o número da possibilidade na tela
        contador += 1
        print(f"Possibilidade {contador}:")
        # Mostrar o quadrado mágico na tela
        print(f"{p[0]} {p[1]} {p[2]}\n{p[3]} {p[4]} {p[5]}\n{p[6]} {p[7]}␣
 ↪{p[8]}\n")

# Mostrar o total de possibilidades encontradas na tela
print(f"Total de possibilidades encontradas: {contador}")
```

```
Possibilidade 1:
2 7 6
9 5 1
4 3 8

Possibilidade 2:
2 9 4
7 5 3
6 1 8

Possibilidade 3:
4 3 8
```

```
9 5 1
2 7 6

Possibilidade 4:
4 9 2
3 5 7
8 1 6

Possibilidade 5:
6 1 8
7 5 3
2 9 4

Possibilidade 6:
6 7 2
1 5 9
8 3 4

Possibilidade 7:
8 1 6
3 5 7
4 9 2

Possibilidade 8:
8 3 4
1 5 9
6 7 2

Total de possibilidades: 8
```