CEFET – UNED Nova Friburgo

Disciplina: Problemas Inversos em Python – 2º semestre 2024

Professora: Josiele da Silva Teixeira

Aluno: Raul Martins Furtado Fernandes

Trabalho Computacional 1

2.1. Escreva uma função que receba um número inteiro e retorne o número invertido.

Código:

```python
def inverte(n):
    # Converte o valor absoluto de 'n' para uma string
    # Obs: a função 'abs' foi utilizada para simplificação do código, mas poderia ser substituída por uma multiplicação por -1 caso o número fosse negativo
    s = str(abs(n))

    # Inicializa a variável 'result' como uma string vazia
    result = ''

    # Inicializa 'i' com o índice do último caractere da string 's'
    i = len(s) - 1

    # Enquanto 'i' for maior ou igual a 0, continua o loop
    while i >= 0:
        # Adiciona o caractere de índice 'i' da string 's' à variável 'result'
        result += s[i]

        # Decrementa 'i' para mover para o próximo caractere à esquerda
        i -= 1

    # Retorna o valor de 'result' convertido de volta para um número inteiro
    return int(result)

# Define o número 'n' para teste
n = 123456789
# Imprime o número original e o resultado invertido
print(f"Numero: {n}")
print(f"Invertido: {inverte(n)}")
```

Output:

```
Numero: 123456789
Invertido: 987654321
```

2.2. Um número é meliante se contém a sequência 171 de dígitos. Ex.: 91713, 5617149. Escreva uma função que receba um número inteiro, identifique e informe ao usuário se esse número é meliante.

Código:

```
def meliante(n):
    # Converte o número 'n' para uma string para facilitar a manipulação dos
dígitos
    s = str(n)

    # Inicializa o índice 'i' como 0
    i = 0

    # Enquanto 'i' for menor que o tamanho da string 's', o loop continua
    while i < len(s):
        # Define os índices 'i1' e 'i2' como i+1 e i+2, respectivamente
        i1 = i + 1
        i2 = i + 2

        # Verifica se 'i1' e 'i2' são índices válidos na string
        if (i1 < len(s)) & (i2 < len(s)):
            # Verifica se a sequência de três dígitos consecutivos é "171"
            if (s[i] == '1') & (s[i1] == '7') & (s[i2] == '1'):
                # Se a sequência for "171", retorna True, indicando que o número
é meliante
                return True

        # Incrementa 'i' para passar para o próximo dígito da string
        i += 1

    # Se nenhuma sequência "171" for encontrada, retorna False
    return False

# Define o número 'n' para testar
n = 123456171789

# Imprime o resultado, informando se o número é meliante ou não
print(f"O numero {n} é meliante? {meliante(n)}")
```

Output:

```
O numero 123456171789 é meliante? True
```

2.3. Um número é legal se não contém dígitos pares e se a soma dos seus dígitos for um número par. Escreva uma função que receba um número inteiro, identifique e informe ao usuário se esse número é legal.

```
def legal(n):
    # Converte o valor absoluto de 'n' para uma string para tratar o número de
forma mais fácil
    # Obs: a função 'abs' foi utilizada para simplificação do código, mas poderia
ser substituída por uma multiplicação por -1 caso o número fosse negativo
    s = str(abs(n))

    # Inicializa a variável 'soma' que irá acumular a soma dos dígitos ímpares
    soma = 0

    # Itera sobre cada caractere da string 's'
    for c in s:
        # Converte o caractere para inteiro
        d = int(c)

        # Verifica se o dígito 'd' é par
        if (d % 2) == 0:
            # Se encontrar um dígito par, retorna False, pois o número não é
"legal"
            return False

        # Se o dígito for ímpar, soma o valor de 'd' à variável 'soma'
        soma += d

    # Após a soma de todos os dígitos ímpares, verifica se a soma é par
    return (soma % 2) == 0

# Define o número 'n' para testar
n = 1135

# Imprime o resultado, informando se o número é "legal" ou não
print(f"O numero {n} é legal? {legal(n)}")
```

Output:

```
O numero 1135 é legal? True
```

2.4. Escreva uma função que receba um número inteiro e faça a divisão de todos os seus dígitos por 2 produzindo um novo número.

```
def divide(n):
    # Converte o valor absoluto de 'n' para uma string para tratar o número de
forma mais fácil
    # Obs: a função 'abs' foi utilizada para simplificação do código, mas poderia
ser substituída por uma multiplicação por -1 caso o número fosse negativo
    s = str(abs(n))

    # Inicializa a variável 'result' como uma string vazia
    result = ''

    # Itera sobre cada caractere da string 's' (cada dígito do número)
    for c in s:
        # Converte o caractere 'c' para um número inteiro
        d = int(c)

        # Divide o dígito 'd' por 2 e adiciona o resultado à string 'result'
        # A operação 'd // 2' realiza uma divisão inteira, descartando a parte
decimal
        result += str(d // 2)

    # Converte a string 'result' de volta para um número inteiro e retorna
    return int(result)

# Define o número 'n' para testar
n = 4712

# Imprime o número original
print(f"Numero: {n}")

# Imprime o resultado da divisão de cada dígito de 'n' por 2
print(f"Dividido: {divide(n)}")
```

Output:

```
Numero: 4712
Dividido: 2301
```

2.5. Escreva uma função para imprimir todos os múltiplos de 5 menores que um número inteiro positivo dado.

```
def multiplosDe5MenoresQue(max):
    # Inicializa a variável 'n' com o valor 0, que será utilizado para armazenar os múltiplos de 5
    n = 0

    # Enquanto 'n' for menor que o valor máximo fornecido, o loop continuará
    while n < max:
        # Imprime o valor atual de 'n', que é um múltiplo de 5
        print(n)

        # Incrementa 'n' por 5 a cada iteração, assim gerando o próximo múltiplo de 5
        n += 5

# Define o valor máximo para a função
n = 20

# Imprime a mensagem informando o valor máximo utilizado
print(f"Múltiplos de 5 menores que {n}:")

# Chama a função para imprimir os múltiplos de 5 menores que o valor de 'n'
multiplosDe5MenoresQue(n)
```

Output:

```
Múltiplos de 5 menores que 20:
0
5
10
15
```

2.6. Escreva um programa que leia os elementos de uma matriz quadrada e em seguida imprima os elementos da diagonal principal.

```
def diagonalPrincipal(matriz):
    # Obtém o tamanho da matriz, ou seja, o número de linhas (assumindo que a
matriz é quadrada)
    size = len(matriz)

    # Inicializa a variável 'i' com 0, que será usada para acessar os elementos
da diagonal
    i = 0

    # Enquanto 'i' for menor que o tamanho da matriz, o loop continua
    while i < size:
        # Imprime o elemento da diagonal principal da matriz na posição (i, i)
        print(f"Elemento {i}x{i}: {matriz[i][i]}")

        # Incrementa 'i' para passar para o próximo elemento da diagonal
        i += 1

# Define uma matriz quadrada 4x4
matriz = [
    [1, 2, 3, 8],
    [4, 5, 6, 6],
    [7, 8, 9, 4],
    [7, 8, 9, 6]
]

# Imprime a mensagem indicando que os elementos da diagonal principal serão
mostrados
print("Elementos da diagonal principal:")

# Chama a função para imprimir os elementos da diagonal principal
diagonalPrincipal(matriz)
```

Output:

```
Elementos da diagonal principal:
Elemento 0x0: 1
Elemento 1x1: 5
Elemento 2x2: 9
Elemento 3x3: 6
```

2.7. Escreva um programa que leia os elementos de uma matriz e em seguida imprima a soma dos elementos de cada coluna.

```
def somaColunas(matriz):
    # Obtém o número de linhas e colunas da matriz
    linhas = len(matriz)
    colunas = len(matriz[0])

    # Inicializa as variáveis 'i' e 'j' que serão usadas para percorrer as linhas
e as colunas
    i = 0
    j = 0

    # Enquanto 'j' for menor que o número de colunas, o loop continua
    while j < colunas:
        # Inicializa a variável 'somaColuna'
        somaColuna = 0

        # Loop para percorrer todas as linhas da coluna 'j'
        while i < linhas:
            # Soma o elemento atual da coluna 'j'
            somaColuna += matriz[i][j]
            # Incrementa 'i' para passar para a próxima linha
            i += 1

        # Imprime a soma dos elementos da coluna 'j'
        print(f"Soma da coluna {j}: {somaColuna}")

        # Reinicia 'i' para 0 e incrementa 'j' para passar para a próxima coluna
        i = 0
        j += 1

# Define uma matriz 2x4 como exemplo
matriz = [ [1, 2, 3, 4], [1, 2, 3, 4] ]

print(f"Soma elementos da coluna:")
# Chama a função para calcular e imprimir a soma das colunas da matriz
somaColunas(matriz)
```

Output:

```
Soma elementos da coluna:
Soma da coluna 0: 2
Soma da coluna 1: 4
Soma da coluna 2: 6
Soma da coluna 3: 8
```

2.8. Escreva uma função para ordenar (crescente) um vetor recebido como argumento de entrada em seguida imprima esse vetor ordenado.

```
# Função de ordenação Insertion Sort
def insertionSort(arr):
    # Inicia o índice 'i' em 1, porque o primeiro elemento já é considerado
ordenado
    i = 1

    # Enquanto 'i' for menor que o tamanho do array, o loop continua
    while i < len(arr):
        # 'j' começa em 'i' e vai retrocedendo enquanto o valor da célula
anterior for maior que o valor atual
        j = i
        # Loop interno para verificar e mover os elementos maiores que arr[j]
para a direita
        while j > 0:
            # Se o elemento anterior for maior que o atual, faz a troca
            if (arr[j - 1] > arr[j]):
                aux = arr[j - 1]
                arr[j - 1] = arr[j]
                arr[j] = aux
                # print(arr)  # Se quiser ver a evolução do array, pode
descomentar
            # Move o índice 'j' para a esquerda
            j -= 1
        # Move o índice 'i' para a próxima posição
        i += 1

    # Retorna o array ordenado
    return arr

# Função de ordenação Selection Sort
def selectionSort(arr):
    # Inicializa o índice 'i' como 0
    i = 0
    # Enquanto 'i' for menor que o tamanho do array, o loop continua
    while i < len(arr):
        # Inicializa o índice 'j' como 'i + 1'
        j = i + 1
        # Loop para percorrer os elementos à direita de arr[i]
        while j < len(arr):
            # Se o elemento atual for maior que o próximo, realiza a troca
            if (arr[i] > arr[j]):
                aux = arr[i]
```

```
                arr[i] = arr[j]
                arr[j] = aux
                # print(arr)  # Se quiser ver a evolução do array, pode
descomentar
            j += 1
        # Move o índice 'i' para a próxima posição
        i += 1
    # Retorna o array ordenado
    return arr

# Array original para ordenação
arr = [5, 2, 8, 9, 4, 3, 7, 0, 1, 6]

# Exibe o array original e a ordenação com Insertion Sort
print(f"Array: {arr}")
print(f"Insertion Sort: {insertionSort(arr)}")

# Outro array para ordenação com Selection Sort
arr2 = [5, 2, 8, 9, 4, 3, 7, 0, 1, 6]

# Exibe o array original e a ordenação com Selection Sort
print(f"Array: {arr2}")
print(f"Selection Sort: {selectionSort(arr2)}")
```

Output:

```
Array: [5, 2, 8, 9, 4, 3, 7, 0, 1, 6]
Insertion Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Array: [5, 2, 8, 9, 4, 3, 7, 0, 1, 6]
Selection Sort: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.9. Escreva uma função que receba um vetor e determine se a soma de todos os seus elementos é maior do que a multiplicação de seus elementos de índice par.

```
def validar(arr):
    # Inicializa as variaveis soma e multiplicacao
    soma = 0
    multiplicacao = 1

    # Inicializa o índice 'i' para percorrer o array
    i = 0
    # Laço para percorrer todos os elementos do array
    while i < len(arr):
        # Armazena o valor do elemento atual
        n = arr[i]

        # Soma o valor do elemento à variável soma
        soma += n

        # Verifica se o índice 'i' é par
        if i % 2 == 0:
            # Se for par, multiplica o valor de 'n' à variável multiplicacao
            multiplicacao *= n

        # Incrementa o índice 'i' para continuar a iteração
        i += 1

    # Exibe a soma de todos os elementos e o produto dos elementos de índice par
    print(f"Soma dos elementos: {soma} | Multiplicação dos elementos de índice
par: {multiplicacao}")

    # Retorna True se a soma for maior que a multiplicação, caso contrário,
retorna False
    return soma > multiplicacao

# Exemplo de array para a função
arr = [2, 8, 6, 9, 5]

# Exibe o resultado da função
print(f"A soma dos elementos do array {arr} é maior do que a multiplicação dos
elementos de índice par? {validar(arr)}")
```

Output:

```
Soma dos elementos: 30 | Multiplicação dos elementos de índice par: 60
A soma dos elementos do array [2, 8, 6, 9, 5] é maior do que a multiplicação dos elementos de índice par? False
```

2.10. Escreva uma função recursiva para identificar se todos os dígitos de um número inteiro são iguais.

```
def validar(n):
    # Converte o valor absoluto de 'n' para uma string para tratar o número de
forma mais fácil
    # Obs: a função 'abs' foi utilizada para simplificação do código, mas poderia
ser substituída por uma multiplicação por -1 caso o número fosse negativo
    s = str(abs(n))

    # Se o número tiver apenas um dígito, todos os dígitos são iguais (pois há só
um)
    if len(s) <= 1:
        return True

    # Se o primeiro dígito for diferente do segundo, retorna False
    if s[0] != s[1]:
        return False

    # Chama recursivamente a função, removendo o primeiro dígito e verificando o
resto dos dígitos
    return validar(int(s[1:]))  # s[1:] retorna todos os elementos menos o
primeiro

# Teste com o número 12345
n = 12345
print(f"Digitos do numero {n} são iguais? {validar(n)}")

# Teste com o número 22222
n = 22222
print(f"Digitos do numero {n} são iguais? {validar(n)}")
```

Output:

```
Digitos do numero 12345 são iguais? False
Digitos do numero 22222 são iguais? True
```

2.11. Escreva uma função com e sem recursividade para retornar o maior elemento de um vetor.

```python
def maior(arr):
    # Inicializa a variável 'maior' com o primeiro elemento da lista
    maior = arr[0]

    # Percorre cada elemento 'n' na lista 'arr'
    for n in arr:
        # Se o elemento 'n' for maior que o 'maior' atual, atualiza o valor de
'maior'
        if (n > maior):
            maior = n

    # Retorna o maior valor encontrado na lista
    return maior

def maiorComRecursividade(arr):
    # Caso base: se a lista tiver apenas um elemento, esse é o maior valor
    if len(arr) == 1:
        return arr[0]

    # Chamada recursiva para encontrar o maior valor na sublista a partir do
segundo elemento
    maiorRestante = maiorComRecursividade(arr[1:])

    # Exibe qual é o maior número encontrado na sublista atual
    # print(f"Maior {arr[1:]}: {maiorRestante}")

    # Compara o primeiro elemento da lista com o maior valor da sublista restante
    # Retorna o maior valor entre os dois
    return arr[0] if arr[0] > maiorRestante else maiorRestante

# Exemplo de lista de números
arr = [2, 81, 6, 9, 5]

print(f"O maior elemento do array {arr} é: {maior(arr)}")
print(f"O maior elemento do array {arr} é: {maiorComRecursividade(arr)}")
```

Output:

```
O maior elemento do array [2, 81, 6, 9, 5] é: 81
O maior elemento do array [2, 81, 6, 9, 5] é: 81
```

2.12. Escreva uma função com e sem recursividade para retornar o total de dígitos de um número inteiro.

```
def digitos(n):
    # Garante que o número é positivo para contar os dígitos
    # Obs: a função 'abs' foi utilizada para simplificação do código, mas poderia
ser substituída por uma multiplicação por -1 caso o número fosse negativo
    n = abs(n)

    # Inicializa a quantidade de dígitos como 1 (pelo menos um dígito)
    digitos = 1

    # Enquanto o número for maior ou igual a 10, divide-o por 10 e aumenta a
contagem de dígitos
    while n >= 10:
        n //= 10  # Divisão inteira por 10
        digitos += 1  # Aumenta o contador de dígitos

    # Retorna o número de dígitos encontrados
    return digitos
def digitosComRecursividade(n):
    # Garante que o número é positivo para contar os dígitos
    # Obs: a função 'abs' foi utilizada para simplificação do código, mas poderia
ser substituída por uma multiplicação por -1 caso o número fosse negativo
    n = abs(n)

    # Caso base: se o número for menor que 10, retorna 1 (apenas um dígito)
    if n < 10:
        return 1

    # Chamada recursiva: divide o número por 10 e adiciona 1 ao número de dígitos
    return 1 + digitosComRecursividade(n // 10)

# Testando para o número 64885
n = 64885

print(f"A quantidade de dígitos do número {n} é: {digitos(n)}")  # Função
Iterativa
print(f"A quantidade de dígitos do número {n} é:
{digitosComRecursividade(n)}")  # Função Recursiva
```

Output:

```
A quantidade de dígitos do número 64885 é: 5
A quantidade de dígitos do número 64885 é: 5
```

2.13. Implemente o método de ordenação Bubble sort.

```python
def bubbleSort(arr):
    # Inicializa a variável 'trocou' para garantir que o loop continue enquanto ocorrerem trocas
    trocou = True
    while(trocou):
        # A cada iteração, considera que não houve trocas (trocou = False)
        trocou = False
        # Itera sobre os elementos da lista
        i = 0
        while i < (len(arr) - 1):
            j = i + 1
            # Se o elemento da posição i for maior que o da posição j, troca-os
            if (arr[i] > arr[j]):
                trocou = True
                aux = arr[i]
                arr[i] = arr[j]
                arr[j] = aux
                # print(arr)  # Descomente para ver o array em cada troca
            i += 1
    # Retorna a lista ordenada
    return arr

# Testando o algoritmo Bubble Sort
arr = [5, 2, 8, 9, 4, 3, 7, 0, 1, 6]

print(f"Array: {arr}")
print(f"Ordenado: {bubbleSort(arr)}")
```

Output:

```
Array: [5, 2, 8, 9, 4, 3, 7, 0, 1, 6]
Ordenado: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.14. Implemente o método de ordenação Bucket sort.

```
def bucketSort(arr, qtd_buckets = 0):
    # Exibe o array que será ordenado
    # print(f"Ordenando array: {arr}")

    # Caso o array tenha 1 ou 0 elementos, ele já está ordenado
    if len(arr) <= 1:
        return arr

    # Caso o array tenha apenas dois elementos, verifica se precisam ser trocados
    if len(arr) == 2:
        if arr[0] > arr[1]:
            # Troca os elementos se estiverem fora de ordem
            aux = arr[0]
            arr[0] = arr[1]
            arr[1] = aux
        return arr

    # Inicializa as variáveis min e max com o primeiro elemento do array
    min = arr[0]
    max = arr[0]
    # Cria uma lista vazia para os buckets
    buckets = []

    # Percorre o array para encontrar os valores mínimo e máximo
    i = 0
    while i < len(arr):
        if arr[i] < min:
            min = arr[i]

        if arr[i] > max:
            max = arr[i]
        i += 1

    # Se os valores mínimo e máximo forem iguais, não há necessidade de ordenar
    if max == min:
        return arr

    # Se qtd_buckets não for fornecido, define o número de buckets como o tamanho
do array
    if qtd_buckets == 0:
        qtd_buckets = len(arr)

    # Cria a quantidade de buckets necessária (lista de listas vazias)
```

```
    i = 0
    while i < qtd_buckets:
        buckets.append([])
        i += 1

    # Calcula o intervalo (step) entre os buckets
    step = (max - min) / qtd_buckets

    # Distribui os elementos nos buckets com base no intervalo (step)
    for n in arr:
        # Calcula o índice do bucket onde o número será inserido
        bkt_i = (n - min) // step

        # Garantir que o índice do bucket não ultrapasse a quantidade de buckets
        if bkt_i >= qtd_buckets:
            bkt_i -= 1

        # Adiciona o número ao bucket correspondente
        buckets[int(bkt_i)].append(n)

    # Exibe os buckets formados
    # print(f"buckets: {buckets}")

    # Ordena recursivamente cada bucket individualmente
    i = 0
    while i < qtd_buckets:
        # A recursão acontece aqui: chama bucketSort para ordenar os buckets
        # O parâmetro qtd_buckets é 0 para que o algoritmo use a quantidade
original de buckets
        buckets[i] = bucketSort(buckets[i],  0 if qtd_buckets == len(arr) else
qtd_buckets)
        i += 1

    # Combina os elementos de todos os buckets ordenados em um array final
    result = []
    for bucket in buckets:
        for n in bucket:
            result.append(n)

    # Retorna o array ordenado
    return result


# Exemplo de uso da função bucketSort
arr = [5, -2.2, 4.9, -1.5, 99, -4, 3.2, -3.5, 1.7, -1.8, 1.6, 1.5]
```

```
# Exibe o array original e o array ordenado após a aplicação do algoritmo
print(f"Array: {arr}")
print(f"Ordenado: {bucketSort(arr, 4)}")
```

Output:

```
Array: [5, -2.2, 4.9, -1.5, 99, -4, 3.2, -3.5, 1.7, -1.8, 1.6, 1.5]
Ordenado: [-4, -3.5, -2.2, -1.8, -1.5, 1.5, 1.6, 1.7, 3.2, 4.9, 5, 99]
```