FACULDADE GRAN TIETÊ
IERT- INSTITUICOES DE ENSINO REUNIDAS TIETE LTDA
Av. 15 de Novembro, 125 – Centro
Barra Bonita - SP

PROGRAMAÇÃO PYTHON

4.º - 10.ª Engenharia da Computação

REVISÃO E TRABALHO 2

Link para envio dos códigos:

https://www.dropbox.com/request/RC4wgYooxvtWeuabeV7j

1. LISTAS EM PYTHON

Listas em Python são estruturas de dados que permitem armazenar coleções ordenadas de elementos. Cada elemento em uma lista pode ser de um tipo de dado diferente, como números inteiros, números de ponto flutuante, strings e até mesmo outras listas. Listas são mutáveis, o que significa que você pode adicionar, remover ou modificar elementos depois de criá-las.

Aqui estão algumas operações comuns que você pode realizar em listas em Python:

A. Criando uma Lista:

Para criar uma lista em Python, você pode usar colchetes [] e adicionar elementos separados por vírgula.

Exemplo:

```
minha_lista = [1, 2, 3, 4, 5]
```

B. Acessando Elementos:

Você pode acessar elementos individuais em uma lista usando índices, começando com 0 para o primeiro elemento.

Exemplo:

```
primeiro_elemento = minha_lista[0]
```

C. Modificando Elementos:

Você pode modificar elementos em uma lista atribuindo um novo valor a um índice específico.

Exemplo:

```
minha_lista[2] = 100
```

D. Adicionando Elementos:

Para adicionar elementos ao final de uma lista, você pode usar o método append().

Exemplo:

```
minha_lista.append(6)
```

E. Removendo Elementos:

Você pode remover elementos de uma lista usando o método remove() ou a função del.

Exemplo:

```
minha_lista.remove(4)  # Remove o elemento 4
del minha_lista[0]     # Remove o primeiro elemento
```

F. Tamanho da Lista:

Você pode verificar o tamanho de uma lista usando a função len().

Exemplo:

```
tamanho = len(minha_lista)  # Retorna o tamanho da lista
```

G. Percorrendo uma Lista:

Você pode usar um loop for para percorrer todos os elementos de uma lista.

Exemplo:

```
for elemento in minha_lista:
    print(elemento)
```

H. Listas Aninhadas:

Você pode criar listas que contêm outras listas, formando listas aninhadas.

Exemplo:

lista_aninhada = [[1, 2, 3], [4, 5, 6]]

Essas são algumas das operações mais comuns que você pode realizar com listas em Python. As listas são uma estrutura de dados versátil e amplamente utilizada em programação Python.

Exercício 1: Contagem de Palavras

Crie um programa que solicita ao usuário uma frase e conta quantas vezes cada palavra aparece na frase. Em seguida, exiba a contagem de cada palavra em ordem alfabética.

```
frase = input("Digite uma frase: ")
palavras = frase.split()  # Divide a frase em palavras

contagem = {}
for palavra in palavras:
    palavra = palavra.lower()  # Converte para letras minúsculas
    if palavra in contagem:
        contagem[palavra] += 1
    else:
        contagem[palavra] = 1

# Ordena e exibe as contagens em ordem alfabética
for palavra, quantidade in sorted(contagem.items()):
    print(f"{palavra}: {quantidade} vezes")
```

Exercício 2: Matriz de Multiplicação

Escreva um programa que cria uma matriz de multiplicação (uma lista de listas) para números de 1 a 5. A matriz deve representar as tabuadas de 1 a 5, e cada sublista deve conter os resultados da multiplicação para um número específico.

```
matriz_multiplicacao = []

for i in range(1, 6):
    linha = []
    for j in range(1, 6):
        resultado = i * j
        linha.append(resultado)
    matriz_multiplicacao.append(linha)

# Exibe a matriz de multiplicação
for linha in matriz_multiplicacao:
    print(linha)
```

Exercício 3: Classificação de Números

Crie um programa que solicita ao usuário uma lista de números e, em seguida, classifica os números em ordem crescente e exibe a lista classificada.

```
entrada = input("Digite uma lista de números separados por espaço: ")
numeros = [float(numero) for numero in entrada.split()]

numeros.sort()  # Classifica a lista em ordem crescente

print("Lista classificada em ordem crescente:")
print(numeros)
```

Exercício 4: Verificação de Anagramas

Desenvolva um programa que verifica se duas palavras inseridas pelo usuário são anagramas uma da outra. Um anagrama é uma palavra ou frase formada pela rearrumação das letras de outra.

```python
palavra1 = input("Digite a primeira palavra: ")
palavra2 = input("Digite a segunda palavra: ")

# Remove espaços em branco e converte para letras minúsculas para simplificar a comparação
palavra1 = palavra1.replace(" ", "").lower()
palavra2 = palavra2.replace(" ", "").lower()

if sorted(palavra1) == sorted(palavra2):
    print("As palavras são anagramas!")
else:
    print("As palavras não são anagramas.")
```

Exercício 5: Estatísticas de Notas

Crie um programa que solicita ao usuário que insira as notas de vários alunos e armazene essas notas em uma lista. Em seguida, calcule a média, a maior e a menor nota, e exiba essas estatísticas.

```python
notas = []
numero_alunos = int(input("Digite o número de alunos: "))

for i in range(numero_alunos):
    nota = float(input(f"Digite a nota do aluno {i + 1}: "))
    notas.append(nota)

media = sum(notas) / len(notas)
nota_maxima = max(notas)
nota_minima = min(notas)

print(f"Média das notas: {media:.2f}")
print(f"Maior nota: {nota_maxima}")
print(f"Menor nota: {nota_minima}")
```

Exercício 6: Contagem de Duplicatas

Escreva um programa que solicita ao usuário uma lista de números e conta quantas vezes cada número duplicado aparece na lista. Em seguida, exiba a contagem de cada número duplicado.

```python
# Solicita uma lista de números separados por espaço
entrada = input("Digite uma lista de números separados por espaço: ")
numeros = [int(numero) for numero in entrada.split()]

contagem = {}
for numero in numeros:
    if numeros.count(numero) > 1:
        contagem[numero] = numeros.count(numero)

# Exibe a contagem de números duplicados
for numero, quantidade in contagem.items():
    print(f"{numero} aparece {quantidade} vezes.")
```

Exercício 7: Média das Extremidades

Crie um programa que recebe uma lista de números e calcula a média dos dois maiores números e dos dois menores números da lista.

```python
# Solicita uma lista de números separados por espaço
entrada = input("Digite uma lista de números separados por espaço: ")
numeros = [float(numero) for numero in entrada.split()]

numeros.sort()
menores = numeros[:2]
maiores = numeros[-2:]

media_menores = sum(menores) / len(menores)
media_maiores = sum(maiores) / len(maiores)

print(f"Média dos dois menores números: {media_menores:.2f}")
print(f"Média dos dois maiores números: {media_maiores:.2f}")
```

Exercício 8: Palavras Únicas

Crie um programa que solicita ao usuário uma frase e exibe as palavras únicas (sem repetições) em ordem alfabética.

```
frase = input("Digite uma frase: ")
palavras = frase.split()
palavras_unicas = list(set(palavras))
palavras_unicas.sort()

print("Palavras únicas em ordem alfabética:")
print(" ".join(palavras_unicas))
```

Exercício 9: Soma de Elementos Únicos

Escreva um programa que recebe uma lista de números e calcula a soma dos números que aparecem apenas uma vez na lista (ou seja, não têm duplicatas).

```
# Solicita uma lista de números separados por espaço
entrada = input("Digite uma lista de números separados por espaço: ")
numeros = [int(numero) for numero in entrada.split()]

soma = 0
for numero in numeros:
    if numeros.count(numero) == 1:
        soma += numero

print("Soma dos números únicos na lista:", soma)
```

Exercício 10: Palíndromo de Palavras

Crie um programa que verifica se uma lista de palavras inserida pelo usuário é um palíndromo. Um palíndromo é uma palavra que é igual quando lida da esquerda para a direita e da direita para a esquerda.

```python
# Solicita uma lista de palavras separadas por espaço
entrada = input("Digite uma lista de palavras separadas por espaço: ")
palavras = entrada.split()

def e_palindromo(palavra):
    return palavra == palavra[::-1]

for palavra in palavras:
    if e_palindromo(palavra):
        print(f"{palavra} é um palíndromo.")
    else:
        print(f"{palavra} não é um palíndromo.")
```

2. TUPLAS

Tuplas em Python são estruturas de dados semelhantes a listas, mas com uma diferença fundamental: as tuplas são imutáveis, o que significa que seus elementos não podem ser alterados após a criação da tupla. As tuplas são definidas utilizando parênteses (), ou simplesmente com elementos separados por vírgula, e podem conter elementos de diferentes tipos de dados.

Aqui estão algumas operações comuns que você pode realizar com tuplas em Python:

A. Criando uma Tupla:

Para criar uma tupla, você pode usar parênteses () ou simplesmente elementos separados por vírgula.

Exemplo:

minha_tupla = (1, 2, 3)

outra_tupla = "a", "b", "c"

B. Acessando Elementos:

Você pode acessar elementos individuais em uma tupla usando índices, da mesma forma que em uma lista.

Exemplo:

```
primeiro_elemento = minha_tupla[0]
```

C. Desempacotamento de Tuplas:

Você pode atribuir os elementos de uma tupla a variáveis individuais em uma única linha.

Exemplo:

```
a, b, c = minha_tupla
```

D. Imutabilidade:

As tuplas são imutáveis, o que significa que você não pode modificar seus elementos após a criação. Isso é diferente das listas, que são mutáveis.

Exemplo:

```
minha_tupla[0] = 10  # Isso resultará em um erro, pois as tuplas são imutáveis
```

E. Concatenação de Tuplas:

Você pode concatenar duas ou mais tuplas para criar uma nova tupla.

Exemplo:

```
tupla1 = (1, 2, 3)
tupla2 = ("a", "b", "c")
nova_tupla = tupla1 + tupla2
```

F. Verificação de Existência de Elemento:

Você pode verificar se um elemento existe em uma tupla usando o operador in.

Exemplo:

```
if 2 in minha_tupla:
   print("O elemento 2 está na tupla.")
```

G. Comprimento da Tupla:

Você pode verificar o tamanho de uma tupla usando a função len().

Exemplo:

tamanho = len(minha_tupla) # Retorna o tamanho da tupla

As tuplas são frequentemente usadas quando você quer garantir que os dados não sejam modificados acidentalmente após sua criação. Elas são úteis para representar conjuntos imutáveis de informações, como coordenadas (latitude e longitude), informações de data/hora, etc.

Exercício 11: Média de Notas

Crie um programa que recebe uma lista de tuplas, onde cada tupla contém o nome de um aluno e suas notas (uma ou mais). Calcule a média das notas de cada aluno e exiba o nome do aluno e sua média.

```
alunos_notas = [("João", (8, 7, 9)), ("Maria", (9, 9, 8)), ("Carlos", (7, 6, 7))]

for aluno, notas in alunos_notas:
    media = sum(notas) / len(notas)
    print(f"{aluno}: Média = {media:.2f}")
```

Exercício 12: Estoque de Produtos

Crie um programa que gerencie um estoque de produtos usando tuplas. Cada tupla deve representar um produto com nome, preço e quantidade em estoque. O programa deve permitir ao usuário adicionar produtos, listar produtos e calcular o valor total do estoque.

```python
estoque = []
total_estoque = 0

while True:
    opcao = input("Escolha uma opção (adicionar/listar/sair): ")

    if opcao == "sair":
        break
    elif opcao == "adicionar":
        nome = input("Digite o nome do produto: ")
        preco = float(input("Digite o preço do produto: "))
        quantidade = int(input("Digite a quantidade em estoque: "))
        produto = (nome, preco, quantidade)
        estoque.append(produto)
        total_estoque += (preco * quantidade)
    elif opcao == "listar":
        for nome, preco, quantidade in estoque:
            print(f"Produto: {nome}, Preço: {preco}, Quantidade: {quantidade}")
        print(f"Valor total do estoque: {total_estoque:.2f}")
```

Exercício 13: Ordenação de Tuplas

Crie um programa que recebe uma lista de tuplas, onde cada tupla contém o nome de uma pessoa e sua idade. Ordene a lista de pessoas por idade em ordem crescente e exiba o resultado.

```python
pessoas_idade = [("Ana", 30), ("João", 25), ("Maria", 35)]

pessoas_idade.sort(key=lambda x: x[1])  # Ordena por idade

for pessoa, idade in pessoas_idade:
    print(f"{pessoa}: {idade} anos")
```

Exercício 14: Contagem de Elementos

Escreva um programa que recebe uma lista de elementos e conta quantas vezes cada elemento aparece na lista. Em seguida, exiba a contagem de cada elemento.

```python
elementos = [1, 2, 2, 3, 4, 4, 4, 5, 5]

contagem = {}
for elemento in elementos:
    if elemento in contagem:
        contagem[elemento] += 1
    else:
        contagem[elemento] = 1

for elemento, quantidade in contagem.items():
    print(f"{elemento}: {quantidade} vezes")
```

Exercício 15: Unindo Duas Listas em Tuplas

Crie um programa que recebe duas listas de elementos e combina essas listas em uma lista de tuplas. Cada tupla deve conter um elemento de cada lista. Em seguida, exiba as tuplas resultantes.

```python
lista1 = ["a", "b", "c"]
lista2 = [1, 2, 3]

tuplas_combinadas = [(elem1, elem2) for elem1, elem2) for elem1, elem2 in zip(lista1, lista2)]

for tupla in tuplas_combinadas:              das:
    print(tupla)
```

3. DICIONÁRIOS

Dicionários em Python são estruturas de dados que permitem armazenar pares de chave-valor. Cada elemento em um dicionário é representado por um par de chave e valor, onde a chave é única e é usada para acessar o valor associado. Dicionários são mutáveis, o que significa que você pode adicionar, modificar ou remover pares chave-valor após a criação do dicionário. As chaves em um dicionário são geralmente strings, mas podem ser de qualquer tipo imutável, como números inteiros ou tuplas.

Aqui estão algumas operações comuns que você pode realizar com dicionários em Python:

A. Criando um Dicionário:

Para criar um dicionário em Python, você pode usar chaves {} e adicionar pares chave-valor separados por dois pontos :.

Exemplo:

meu_dicionário = {"nome": "João", "idade": 30, "cidade": "São Paulo"}

B. Acessando Valores:

Você pode acessar o valor associado a uma chave específica em um dicionário usando a chave.

Exemplo:

nome = meu_dicionário["nome"]

C. Modificando Valores:

Você pode modificar o valor associado a uma chave em um dicionário atribuindo um novo valor a essa chave.

Exemplo:

meu_dicionário["idade"] = 31

D. Adicionando Pares Chave-Valor:

Para adicionar um novo par chave-valor a um dicionário, você pode simplesmente atribuir um valor a uma chave que ainda não existe no dicionário.

Exemplo:

meu_dicionário["profissão"] = "Engenheiro"

E. Removendo Pares Chave-Valor:

Você pode remover um par chave-valor de um dicionário usando a palavra-chave del.

Exemplo:

del meu_dicionário["cidade"]

F. Verificação de Existência de Chave:

Você pode verificar se uma chave existe em um dicionário usando o operador in.

Exemplo:

if "idade" in meu_dicionário:

print("A chave 'idade' existe no dicionário.")

G. Chaves, Valores e Itens:

Você pode obter uma lista de todas as chaves, valores ou pares chave-valor de um dicionário usando os métodos keys(), values() e items(), respectivamente.

Exemplo:

chaves = meu_dicionário.keys()

valores = meu_dicionário.values()

itens = meu_dicionário.items()

Dicionários são amplamente utilizados em Python para mapear informações, armazenar configurações, criar estruturas de dados complexas e muito mais. Eles oferecem uma maneira eficiente de acessar e manipular dados com base em chaves únicas.

Exercício 16: Contagem de Palavras

Crie um programa que solicita ao usuário uma frase e conta quantas vezes cada palavra aparece na frase. Em seguida, exiba a contagem de cada palavra.

```
frase = input("Digite uma frase: ")
palavras = frase.split()
contagem = {}

for palavra in palavras:
    palavra = palavra.lower()  # Converte para letras minúsculas
    if palavra in contagem:
        contagem[palavra] += 1
    else:
        contagem[palavra] = 1

for palavra, quantidade in contagem.items():
    print(f"'{palavra}': {quantidade} vezes")
```

Exercício 17: Cadastro de Alunos

Crie um programa que permite ao usuário cadastrar alunos associando seus nomes a notas. O programa deve permitir a adição de novos alunos, consulta de notas e cálculo da média das notas dos alunos.

```
alunos = {}

while True:
    opcao = input("Escolha uma opção (adicionar/consultar/media/sair): ")

    if opcao == "sair":
        break
    elif opcao == "adicionar":
        nome = input("Digite o nome do aluno: ")
        nota = float(input("Digite a nota do aluno: "))
        alunos[nome] = nota
    elif opcao == "consultar":
        nome = input("Digite o nome do aluno para consultar a nota: ")
        if nome in alunos:
            print(f"A nota de {nome} é {alunos[nome]:.2f}")
        else:
            print(f"{nome} não encontrado.")
    elif opcao == "media":
        if len(alunos) == 0:
            print("Nenhum aluno cadastrado.")
        else:
            media = sum(alunos.values()) / len(alunos)
            print(f"Média das notas dos alunos: {media:.2f}")
```

Exercício 18: Contagem de Letras

Crie um programa que solicita ao usuário uma palavra e conta quantas vezes cada letra aparece na palavra. Em seguida, exiba a contagem de cada letra.

```python
palavra = input("Digite uma palavra: ")
contagem = {}

for letra in palavra:
    letra = letra.lower()  # Converte para letras minúsculas
    if letra in contagem:
        contagem[letra] += 1
    else:
        contagem[letra] = 1

for letra, quantidade in contagem.items():
    print(f"'{letra}': {quantidade} vezes")
```

Exercício 19: Agenda de Contatos

Crie um programa que permite ao usuário gerenciar uma agenda de contatos, onde cada contato é representado pelo nome e pelo número de telefone. O programa deve permitir adicionar novos contatos, consultar números de telefone e listar todos os contatos.

```python
agenda = {}

while True:
    opcao = input("Escolha uma opção (adicionar/consultar/listar/sair): ")

    if opcao == "sair":
        break
    elif opcao == "adicionar":
        nome = input("Digite o nome do contato: ")
        telefone = input("Digite o número de telefone: ")
        agenda[nome] = telefone
    elif opcao == "consultar":
        nome = input("Digite o nome do contato para consultar o telefone: ")
        if nome in agenda:
            print(f"O número de telefone de {nome} é {agenda[nome]}")
        else:
            print(f"{nome} não encontrado.")
    elif opcao == "listar":
        for nome, telefone in agenda.items():
            print(f"{nome}: {telefone}")
```

Exercício 20: Calculadora de Compras

Crie um programa que permite ao usuário registrar compras em uma lista de produtos associados a preços. O programa deve calcular o valor total das compras e exibir a lista de produtos comprados.

```
compras = {}
total = 0

while True:
    produto = input("Digite o nome do produto (ou 'sair' para encerrar): ")

    if produto == "sair":
        break
    elif produto in compras:
        quantidade = int(input("Digite a quantidade comprada: "))
        compras[produto] += quantidade
    else:
        preco = float(input("Digite o preço do produto: "))
        quantidade = int(input("Digite a quantidade comprada: "))
        compras[produto] = quantidade
        total += preco
```

4. PROGRAMAÇÃO ORIENTADA A OBJETOS

Programação Orientada a Objetos (POO) é um paradigma de programação que se baseia no conceito de "objetos". Os objetos são instâncias de classes, que são estruturas que definem as propriedades (atributos) e comportamentos (métodos) que os objetos podem ter. A POO é uma abordagem poderosa para modelar e resolver problemas complexos, pois permite organizar o código de forma mais modular e reutilizável.

A. Classes e Objetos:

Classe: Uma classe é uma estrutura que define um tipo de objeto. Ela define os atributos (variáveis) e métodos (funções) que os objetos dessa classe terão. Classes servem como modelos para a criação de objetos.

Objeto: Um objeto é uma instância de uma classe. Ele possui seus próprios valores de atributos, mas compartilha os métodos definidos na classe.

B. Atributos:

Atributos são variáveis que armazenam dados relacionados a um objeto. Eles representam as características do objeto. Os atributos são definidos na classe e podem ser acessados e modificados pelos métodos da classe.

C. Métodos:

Métodos são funções definidas em uma classe que descrevem os comportamentos dos objetos dessa classe. Eles operam nos atributos do objeto e podem realizar ações específicas.

D. Encapsulamento:

Encapsulamento é o princípio que restringe o acesso direto aos atributos de um objeto e incentiva o uso de métodos para interagir com esses atributos. Em Python, é comum usar métodos "getter" e "setter" para acessar e modificar atributos.

E. Herança:

Herança é um conceito que permite criar uma nova classe com base em uma classe existente. A nova classe herda os atributos e métodos da classe pai (superclasse) e pode adicionar ou modificar funcionalidades.

F. Polimorfismo:

Polimorfismo é a capacidade de diferentes classes responderem ao mesmo método de maneira única. Isso permite tratar objetos de diferentes classes de forma uniforme.

Exercício 21: Classe Carro

Crie uma classe chamada Carro que tenha os atributos marca e modelo. Crie um objeto da classe Carro e exiba a marca e o modelo do carro.

```python
class Carro:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

meu_carro = Carro("Ford", "Fiesta")
print(f"Marca: {meu_carro.marca}, Modelo: {meu_carro.modelo}")
```

Exercício 22: Classe Retângulo

Crie uma classe chamada Retangulo que tenha os atributos largura e altura. Crie um método calcular_area que calcula a área do retângulo (largura * altura).

```python
class Retangulo:
    def __init__(self, largura, altura):
        self.largura = largura
        self.altura = altura

    def calcular_area(self):
        return self.largura * self.altura

meu_retangulo = Retangulo(5, 3)
area = meu_retangulo.calcular_area()
print(f"Área do retângulo: {area}")
```

Exercício 23: Classe Animal

Crie uma classe chamada Animal com um método emitir_som que imprime "Som de animal genérico". Crie subclasses como Cachorro e Gato que sobrescrevem o método emitir_som para imprimir sons específicos de cada animal.

```python
class Animal:
    def emitir_som(self):
        print("Som de animal genérico")

class Cachorro(Animal):
    def emitir_som(self):
        print("Latido")

class Gato(Animal):
    def emitir_som(self):
        print("Miado")

animal1 = Animal()
cachorro1 = Cachorro()
gato1 = Gato()

animal1.emitir_som()  # Saída: Som de animal genérico
cachorro1.emitir_som()  # Saída: Latido
gato1.emitir_som()  # Saída: Miado
```

Exercício 24: Classe Conta Bancária

Crie uma classe chamada ContaBancaria com os atributos saldo e titular. Adicione métodos para depositar e sacar dinheiro da conta. Crie um objeto da classe ContaBancaria e realize algumas operações.

```python
class ContaBancaria:
    def __init__(self, titular, saldo=0):
        self.titular = titular
        self.saldo = saldo

    def depositar(self, valor):
        self.saldo += valor

    def sacar(self, valor):
        if valor <= self.saldo:
            self.saldo -= valor
        else:
            print("Saldo insuficiente.")

conta1 = ContaBancaria("João", 1000)
print(f"Saldo inicial de {conta1.titular}: {conta1.saldo}")
conta1.depositar(500)
print(f"Novo saldo após depósito: {conta1.saldo}")
conta1.sacar(300)
print(f"Novo saldo após saque: {conta1.saldo}")
```

Exercício 25: Classe Livro

Crie uma classe chamada Livro com os atributos titulo e autor. Crie um método mostrar_informacoes que exiba o título e o autor do livro. Crie objetos da classe Livro e chame o método mostrar_informacoes para exibir as informações dos livros.

```python
class Livro:
    def __init__(self, titulo, autor):
        self.titulo = titulo
        self.autor = autor

    def mostrar_informacoes(self):
        print(f"Título: {self.titulo}")
        print(f"Autor: {self.autor}")

livro1 = Livro("Dom Quixote", "Miguel de Cervantes")
livro2 = Livro("A Moreninha", "Joaquim Manuel de Macedo")

livro1.mostrar_informacoes()
print()
livro2.mostrar_informacoes()
```