Programação em Python

CEFET-MG

CEFET-MG

- **Listas**
  - Coleção ordenada de itens.

Programação em Python

CEFET-MG

- **Listas**
  - Coleção ordenada de itens.
  - Qualquer objeto Python.

CEFET-MG

- **Listas**
  - Coleção ordenada de itens.
  - Qualquer objeto Python.
    - Inclusive de listas e tuplas.

- **Listas**
  - Coleção ordenada de itens.
  - Qualquer objeto Python.
    - Inclusive de listas e tuplas.
  - Alocação dinâmica de memória.

Programação em Python

CEFET-MG

- **Listas**
  - Coleção ordenada de itens.
  - Qualquer objeto Python.
    - Inclusive de listas e tuplas.
  - Alocação dinâmica de memória.
  - Indicado pelo ‘[ ]’.

CEFET-MG

- **Listas**
  - Coleção ordenada de itens.
  - Qualquer objeto Python.
    - Inclusive de listas e tuplas.
  - Alocação dinâmica de memória.
  - Indicado pelo ‘[ ]’.
  - Itens separados por ‘,’ (vírgula).

- **Listas**
  - Exemplo:

```
>>> bandas = [‘AcDc’, ‘Black Sabbatah’, ’Metallica’, ’Iron Maiden’ ]
```

- **Listas**
  - Exemplo:

```
>>> bandas = [‘AcDc’, ‘Black Sabbatah’, ’Metallica’, ’Iron Maiden’ ]
```

Dica: use ‘s’ no nome da lista

- **Listas**
  - Exemplo:

```
>>> bandas = [‘AcDc’, ‘Black Sabbatah’, ’Metallica’, ’Iron Maiden’ ]
```

  - **O que acontece se manda imprimir?**

- **Listas**
  - Exemplo:

```
>>> bandas = [‘AcDc’, ‘Black Sabbatah’, ’Metallica’, ’Iron Maiden’ ]
```

  - **O que acontece se manda imprimir?**
    - **Esta é uma ‘boa’ forma de exibição?**

- **Listas**
  - Exemplo:

```
>>> bandas = [‘AcDc’, ‘Black Sabbatah’, ’Metallica’, ’Iron Maiden’ ]
```

  - **O que acontece se manda imprimir?**
    - **Esta é uma ‘boa’ forma de exibição?**
      - Como corrigir: acessar os elementos individuais.

- **Acessando elementos de uma lista**
  - Ordenada:
    - Posição ou índice.

- **Acessando elementos de uma lista**
  - Ordenada:
    - Posição ou índice.
      - bandas[0] **{#Primeira posição é o '0'}**

- **Acessando elementos de uma lista**
  - Ordenada:
    - Posição ou índice.
      - bandas[0] **{#Primeira posição é o '0'}**

```
>>> print(bandas[0])
```

- **Acessando elementos de uma lista**
  - Ordenada:
    - Posição ou índice.
      - bandas[0] **{#Primeira posição é o '0'}**

```
>>> print(bandas[0])

>>> print(bandas[4])
```

- **Acessando elementos de uma lista**
  - Ordenada:
    - Posição ou índice.
      - bandas[0] **{#Primeira posição é o '0'}**

```
>>> print(bandas[0])

>>> print(bandas[4])
```

      - **E se pedirmos a posição '-1' ?**

- **Acessando elementos de uma lista**
  - Ordenada:
    - Posição ou índice.
      - bandas[0] **{#Primeira posição é o '0'}**

```
>>> print(bandas[0])

>>> print(bandas[4])
```

      - **E se pedirmos a posição '-1' ?**
        - Índice negativo: começa a partir do fim da lista.

- **Exercícios:**

1. Nomes: Armazene os nomes de alguns de seus amigos em uma lista chamada names. Exiba o nome de cada pessoa acessando cada elemento da lista, um de cada vez.

- **Exercícios:**

```
names = ["Ana", "Carlos", "João", "Mariana", "Fernanda"]


# Exibindo cada nome acessando diretamente pelo índice
print(names[0])
print(names[1])
print(names[2])
print(names[3])
print(names[4])
```

- **Exercícios:**

  Saudações: Comece com a lista usada no Exercício 1, mas em vez de simplesmente exibir o nome de cada pessoa, apresente uma mensagem a elas. O texto de cada mensagem deve ser o mesmo, porém cada mensagem deve estar personalizada com o nome da pessoa.

- **Exercícios:**

```
# Lista com nomes
names = ["Ana", "Carlos", "João", "Mariana", "Fernanda"]


# Mensagens personalizadas
print(f"{names[0]}, você é uma amiga incrível!")
print(f"{names[1]}, espero que esteja se divertindo hoje.")
print(f"{names[2]}, bom trabalho no seu último projeto!")
print(f"{names[3]}, parabéns pelo seu esforço nos estudos.")
print(f"{names[4]}, continue sempre sorrindo!")
```

- **Exercícios:**

  Sua própria lista: Pense em seu meio de transporte preferido, como motocicleta ou carro, e crie uma lista que armazene vários exemplos. Utilize sua lista para exibir uma série de frases sobre esses itens, como “Gostaria de ter uma moto Honda”.

- **Exercícios:**

```
# Lista com meios de transporte
transportes = ["moto Honda", "carro Tesla", "bicicleta Caloi", "patinete elétrico"]


# Exibindo frases personalizadas
print(f"Gostaria de ter uma {transportes[0]}.")
print(f"Gostaria de dirigir um {transportes[1]}.")
print(f"Gostaria de pedalar uma {transportes[2]}.")
print(f"Gostaria de andar em um {transportes[3]}.")
```

CEFET-MG

- **Comando for**
  - Laço de repetição
  - for *índice* in [*lista*]:

- **Comando for**
  - Laço de repetição
  - for *índice* in [*lista*]:
  - Ex.:

```
>>> for i in [1, 2, 3, 4]:
>>>         print(bandas[i])
```

- **Comando for**
  - Laço de repetição
  - for *índice* in [*lista*]:
  - Ex.:

```
>>> for i in [1, 2, 3, 4]:
>>>         print(bandas[i])
```

  - **Por que está errado?**

Programação em Python

CEFET-MG

- **Comando for**
  - Laço de repetição
  - for *índice* in [*lista*]:
  - Ex.:

```
>>> for i in [0, 1, 2]:
>>>         print(bandas[i])
```

- **Comando for**
  - Laço de repetição
  - for *índice* in [*lista*]:
  - Ex.:

```
>>> for i in [0, 1, 2]:
>>>         print(bandas[i])
```

E se eu não souber o tamanho da lista?

- **Comandos**
  - **len()** Retorna o tamanho da lista
  - range(*start, stop, step*) Retorna um intervalo entre *start* e *stop*, com o passo *step*.
    - range(*stop*) Retorna de 0 a *stop.*
    - range(*start, stop*) Retorna o intervalo entre *start* e *stop*, com passo 1.

- **Comandos**
  - Ex.:

```
>>> for i in range(len(bandas)):
```

Ou

```
>>> tamanho = int (len(bandas))
>>> for i in range(tamanho):
```

Recomendado, mais 'Pythonico'!

- **Comandos**
  - Ex.:

```
for i in range(0, 10, 2):
    print(i)

for i in range(2, 6):
    print(i)

for i in range(5):
    print(i)
```

- **Exercícios:**

1. Nomes: Armazene os nomes de alguns de seus amigos em uma lista chamada names. Exiba o nome de cada pessoa acessando cada elemento da lista, um de cada vez.
2. Saudações: Comece com a lista usada no Exercício 1, mas em vez de simplesmente exibir o nome de cada pessoa, apresente uma mensagem a elas. O texto de cada mensagem deve ser o mesmo, porém cada mensagem deve estar personalizada com o nome da pessoa.
3. Sua própria lista: Pense em seu meio de transporte preferido, como motocicleta ou carro, e crie uma lista que armazene vários exemplos. Utilize sua lista para exibir uma série de frases sobre esses itens, como “Gostaria de ter uma moto Honda”.

- **Exercícios:**

```
# Lista com nomes de amigos
names = ["Ana", "Carlos", "João", "Mariana", "Fernanda"]


# Exibindo cada nome com um loop
for name in names:
    print(name)
```

- **Exercícios:**

```
# Lista com nomes de amigos
names = ["Ana", "Carlos", "João", "Mariana", "Fernanda"]


# Exibindo uma saudação personalizada para cada um
for name in names:
    print(f"Olá, {name}! Espero que você tenha um ótimo dia!")
```

- **Exercícios:**

```
# Lista com meios de transporte
transportes = ["moto Honda", "carro Tesla", "bicicleta Caloi", "patinete elétrico"]

# Usando for para exibir frases personalizadas
for item in transportes:
    print(f"Gostaria de ter um(a) {item}.")
```

- **Outras formas do Comando for**
  - for banda in bandas:
    - print(bandas[banda])

  - palavra = "Python"
  - for letra in palavra:
    - print(letra)

- **E se eu quiser imprimir o índice e o valor?**

- **Comando for**
  - **E se eu quiser imprimir o índice e o valor?**

enumerate()

Ex.:

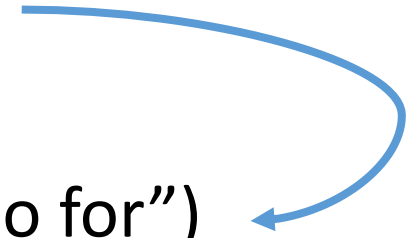
```
frutas = ["maçã", "banana", "cereja"]
for indice, fruta in  enumerate(frutas):
    print(f"Índice {indice}: {fruta}")
```

- **Comando for**
  - **break** Termina o laço for imediatamente.
  - **continue** Pula a iteração atual e passa para a próxima iteração do laço.

- **Comando for**
  - **break** Termina o laço for imediatamente.
  - **continue** Pula a iteração atual e passa para a próxima iteração do laço.

Ex.:

```
for i in range(5):
        if i == 3:
                print("Encontrei 3, saindo do loop.")
                break
        print(i)
print(“Estou for a do for”)
```



- **Comando for**
  - **break** Termina o laço for imediatamente.
  - **continue** Pula a iteração atual e passa para a próxima iteração do laço.

Ex.:

```
for i in range(5):
    if i == 2:
        print("Pulando o 2.")
        continue    # Pula o resto do bloco para i=2
    print(i)
```

- **Comando for**
  - **zip()** usado para iterar sobre múltiplos iteráveis em paralelo.
  - Ex.:

```
nomes = [“Angus", “Malcon", “Bom Scott"]
idades = [25, 30, 22]
for nome, idade in zip(nomes, idades):
        print(f"{nome} tem {idade} anos.")
```

- **Comando for**
  - **else** executado apenas se o laço completar todas as suas iterações.
  - Ex.:

```
for i in range(3):
    print(i)
else:
    print("Loop concluído sem interrupções.")
```

- **Comando for**
  - **else** executado apenas se o laço completar todas as suas iterações.
  - Ex.:

```
for i in range(3):
        if i == 2
                break
        print(i)
else:
        print("Loop concluído sem interrupções.")
```

CEFET-MG

- **Comando for**
  - Para criar listas (Linha única)
  - Sintaxe básica: [instrução for item in iteravel]
  - Com condição: [instrução for item in iteravel if condicao]
  - Ex.:

```
numeros = [1, 2, 3, 4, 5]
quadrados = [numero * numero for numero in numeros]
print(quadrados)
```

CEFET-MG

- **Comando for**
  - Ex.:

```
pares = [numero for numero in range(10) if numero % 2 == 0]
print(pares)

pares_aninhados = [(x, y) for x in range(2) for y in range(2)]
print(pares_aninhados)
```

- **Modificar elementos de uma lista**
  - Acessar o elemento individual.

- **Modificar elementos de uma lista**
  - Acessar o elemento individual.

```
>>>bandas[0]
>>>print(bandas[0])
```

- **Modificar elementos de uma lista**
  - Acessar o elemento individual.

```
>>>bandas[0]
>>>print(bandas[0])
```

  - Inserir novo

- **Modificar elementos de uma lista**
  - Acessar o elemento individual.

    >>>bandas[0]

    >>>print(bandas[0])

  - Inserir novo
    - .append('*novo elemento*')

- **Modificar elementos de uma lista**
  - Acessar o elemento individual.

```
>>>bandas[0]
>>>print(bandas[0])
```

  - Inserir novo
    - .append('*novo elemento*') **{#Insere no fim da lista}**

- **Modificar elementos de uma lista**
  - Acessar o elemento individual.

    >>>bandas[0]

    >>>print(bandas[0])

  - Inserir novo
    - .append(‘*novo elemento*’)

    >>> bandas.append(‘Ana Castela’)

    >>>print(bandas)

- **Modificar elementos de uma lista**
  - Acessar o elemento individual.

    >>>bandas[0]

    >>>print(bandas[0])

  - Inserir novo
    - .append('*novo elemento*')

    >>> bandas.append('Ana Castela')

    >>>print(bandas)

- **E se fizermos >>> bandas[-1] = 'Ana Castela')?**

- **Modificar elementos de uma lista**
  - Inserir em qualquer posição
    - .insert(*índice*, *elemento*)

- **Modificar elementos de uma lista**
  - Inserir em qualquer posição
    - .insert(*índice*, *elemento*)
    - Ex.:

```
>>> bandas.insert(0, 'Raul Seixas')
```

- **Modificar elementos de uma lista**
  - Inserir em qualquer posição
    - .insert(*índice*, *elemento*)
    - Ex.:

```
>>> bandas.insert(0, 'Raul Seixas')

>>> bandas.inset(-2, 'Bêbados e habilidosos')
```

- **Modificar elementos de uma lista**
  - Inserir em qualquer posição
    - .insert(*índice*, *elemento*)
    - Ex.:

```
>>> bandas.insert(0, 'Raul Seixas')

>>> bandas.inset(-2, 'Bêbados e habilidosos')
```

- **O elemento anterior: foi substituído?**

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - del nome_lista[índice]

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - del nome_lista[índice]
    - Ex.:

```
>>> del bandas[-1]
```

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - del nome_lista[índice]
    - Ex.:

```
>>> del bandas[-1]
```

    - **O que aconteceu com a posição dos outros elementos?**

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Lista vista como uma pilha.

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Lista vista como uma pilha.
    - Retira o último elemento, mas permite que seja utilizado.

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Lista vista como uma pilha.
    - Retira o último elemento, mas permite que seja utilizado.
    - .pop()

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Lista vista como uma pilha.
    - Retira o último elemento, mas permite que seja utilizado.
    - .pop()
    - Ex.:

```
>>> retirado =  bandas.pop()
>>> print(bandas)
>>> print(retirado)
```

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - **E se quiser retirar de outra posição?**

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - **E se quiser retirar de outra posição?**
      - .pop*(índice).*

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - **E se quiser retirar de outra posição?**
      - .pop*(índice).*
      - Ex.:

```
>>> retirado = bandas.pop(0)
>>> print(retirado)
>>> print(bandas)
```

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Não sei a posição.

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Não sei a posição.
    - Mas sei o valor do elemento.

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Não sei a posição.
    - Mas sei o valor do elemento.
    - .remove(elemento)

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Não sei a posição.
    - Mas sei o valor do elemento.
    - .remove(elemento)
    - Ex.:

```
>>> elemento = 'AcDc'
>>> bandas.remove(elemento)
>>> print(bandas)
```

CEFET-MG

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Não sei a posição.
    - Mas sei o valor do elemento.
    - .remove(elemento)
    - Ex.:

```
>>> elemento = 'AcDc'
>>> bandas.remove(elemento)
>>> print(bandas)
```

 # Apaga apenas a primeira ocorrência do valor. Se houver mais de um: usar laço.

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Não sei a posição.
    - Mas sei o valor do elemento.

- **Modificar elementos de uma lista**
  - Removendo elementos:
    - Não sei a posição.
    - Mas sei o valor do elemento.
    - .remove(elemento)

 # Apaga apenas a primeira ocorrência do valor.
Se houver mais de um: usar laço.

- **Exercícios:**

7. Lista de convidados: Crie uma lista que inclua pelo menos quatro pessoas que você gostaria de convidar para jantar. Em seguida, utilize sua lista para exibir uma mensagem para cada pessoa, convidando-a para jantar.
8. Alterando a lista de convidados: Você acabou de saber que um de seus convidados não poderá comparecer ao jantar, portanto será necessário enviar um novo conjunto de convites. Você deverá pensar em outra pessoa para convidar.
9. Acrescente uma instrução print no final de seu programa, especificando o nome do convidado que não poderá comparecer.
10. Modifique sua lista, substituindo o nome do convidado que não poderá comparecer pelo nome da nova pessoa que você está convidando.
11. Exiba um segundo conjunto de mensagens com o convite, uma para cada pessoa que continua presente em sua lista.
12. Mais convidados: Você acabou de encontrar uma mesa de jantar maior, portanto agora tem mais espaço disponível. Pense em mais três convidados para o jantar.

Programação em Python

CEFET-MG

```
# Lista inicial de convidados
convidados = ["Ana", "Carlos", "Mariana", "João"]

# Exibindo os convites iniciais
print("Convites iniciais:")
for pessoa in convidados:
    print(f"Olá {pessoa}, gostaria de convidá-lo(a) para um jantar especial.")

# Um convidado não poderá comparecer
nao_vai = "Carlos"
print(f"\nInfelizmente, {nao_vai} não poderá comparecer ao jantar.")

# Substituindo o convidado
novo_convidado = "Fernanda"
convidados[convidados.index(nao_vai)] = novo_convidado

# Exibindo os convites atualizados
print("\nConvites atualizados:")
for pessoa in convidados:
    print(f"Olá {pessoa}, gostaria de convidá-lo(a) para um jantar especial.")

# Encontrou uma mesa maior -> mais três convidados
print("\nBoa notícia! Encontrei uma mesa de jantar maior.")
convidados.insert(0, "Lucas")      # adiciona no início
convidados.insert(2, "Beatriz")    # adiciona no meio
convidados.append("Rafael")        # adiciona no final

# Exibindo a lista final de convites
print("\nConvites finais:")
for pessoa in convidados:
    print(f"Olá {pessoa}, você está convidado(a) para o jantar especial!")
```

CEFET-MG

```
# Lista inicial de convidados
convidados = ["Ana", "Carlos", "Mariana", "João"]

# Exibindo os convites iniciais
print("Convites iniciais:")
for pessoa in convidados:
    print(f"Olá {pessoa}, gostaria de convidá-lo(a) para um jantar especial.")

# Um convidado não poderá comparecer
nao_vai = "Carlos"
print(f"\nInfelizmente, {nao_vai} não poderá comparecer ao jantar.")

# Substituindo o convidado
novo_convidado = "Fernanda"
convidados[convidados.index(nao_vai)] = novo_convidado
```

Programação em Python

CEFET-MG

- # Exibindo os convites atualizados
- print("\nConvites atualizados:")
- for pessoa in convidados:
-     print(f"Olá {pessoa}, gostaria de convidá-lo(a) para um jantar especial.")

- # Encontrou uma mesa maior -> mais três convidados
- print("\nBoa notícia! Encontrei uma mesa de jantar maior.")
- convidados.insert(0, "Lucas") # adiciona no início
- convidados.insert(2, "Beatriz") # adiciona no meio
- convidados.append("Rafael") # adiciona no final

- # Exibindo a lista final de convites
- print("Convites finais:")
- for pessoa in convidados:
-     print(f"Olá {pessoa}, você está convidado(a) para o jantar especial!")

- **Exercícios:**

13. Reduzindo a lista de convidados: Você acabou de descobrir que sua nova mesa de jantar não chegará a tempo para o jantar e tem espaço para somente dois convidados.
14. Comece com seu programa do Exercício 6. Acrescente uma nova linha que mostre uma mensagem informando que você pode convidar apenas duas pessoas para o jantar.
15. Utilize pop() para remover os convidados de sua lista, um de cada vez, até que apenas dois nomes permaneçam em sua lista. Sempre que remover um nome de sua lista, mostre uma mensagem a essa pessoa, permitindo que ela saiba que você sente muito por não poder convidá-la para o jantar.
16. Apresente uma mensagem para cada uma das duas pessoas que continuam na lista, permitindo que elas saibam que ainda estão convidadas.
17. Utilize del para remover os dois últimos nomes de sua lista, de modo que você tenha uma lista vazia. Mostre sua lista para garantir que você realmente tem uma lista vazia no final de seu programa.

```
# Lista inicial (já ampliada do exercício anterior)
convidados = ["Lucas", "Ana", "Beatriz", "Fernanda", "Mariana", "João", "Rafael"]

print("Infelizmente, a nova mesa não chegará a tempo...")
print("Só posso convidar DUAS pessoas para o jantar.")

# Removendo convidados até restarem apenas 2
while len(convidados) > 2:
    removido = convidados.pop()
    print(f"Sinto muito, {removido}, mas não poderei convidá-lo(a) para o jantar.")

# Mensagem para os dois que ainda estão na lista
print("Os convidados que continuam na lista:")
for pessoa in convidados:
    print(f"{pessoa}, você ainda está convidado(a) para o jantar!")

# Limpando a lista completamente
del convidados[:]


# Mostrando lista final para garantir que está vazia

print("Lista final de convidados:", convidados)
```

CEFET-MG

- **Comandos úteis para Listas**
  - ‘+’ Concatenação de listas

CEFET-MG

- **Comandos úteis para Listas**
  - ‘+’ Concatenação de listas
  - Ex.:

```
>>> a = [1, 2, 3]

>>> b = [4, 5, 6]

>>> c = a + b
```

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)
  - lista[início:fim:passo]

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)
  - lista[início:fim:passo]
  - Ex.:

```
>>> t = [9, 41, 12, 3, 74, 15]

>>> t[1:3]
```

CEFET-MG

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)
  - lista[início:fim:passo]
  - Ex.:

```
>>> t = [9, 41, 12, 3, 74, 15]

>>> t[1:3]
```

Quais elementos foram impressos? E o 3?

CEFET-MG

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)
  - lista[início:fim:passo]
  - Ex.:

```
>>> t = [9, 41, 12, 3, 74, 15]

>>> t[1:3]
```

Quais elementos foram impressos? E o 3?

Assim como nas *strings*, o segundo número é “até, mas não incluso”.

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)
  - Ex.:

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


print(numeros[2:6])
        # [2, 3, 4, 5]  (do índice 2 até antes do 6)
print(numeros[:5])
        # [0, 1, 2, 3, 4]  (do início até antes do índice 5)
```

CEFET-MG

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)
  - Ex.:

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


print(numeros[5:])
        # [5, 6, 7, 8, 9]  (do índice 5 até o final)
print(numeros[::2])
        # [0, 2, 4, 6, 8]  (pega de 2 em 2)
```

- **Comandos úteis para Listas**
  - ‘:’ Separação de listas (Fatiamento)
  - Ex.:

```
numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]


print(numeros[::2])
        # [1, 3, 5, 7, 9]  (Passo dois)
print(numeros[::-1])
        # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]  (inverte a lista)
```

- **Comandos úteis para Listas**
  - ‘in’ Verifica se o elemento está na lista

- **Comandos úteis para Listas**
  - ‘in’ Verifica se o elemento está na lista
  - ‘not in’ Verifica se o elemento está na lista

- **Comandos úteis para Listas**
  - ‘in’ Verifica se o elemento está na lista
  - ‘not in’ Verifica se o elemento está na lista
    - Retorna True ou False

- **Comandos úteis para Listas**
  - ‘in’ Verifica se o elemento está na lista
  - ‘not in’ Verifica se o elemento está na lista
    - Retorna True ou False
  - Ex.:

```
>>> nums = [9, 41, 12, 3, 74, 15]
>>> 9 in nums
>>> 9 not in nums
>>> 0 in nums
>>> 0 not in nums
```

- **Comandos úteis para Listas**
  - ‘max’ Retorna o máximo da lista

- **Comandos úteis para Listas**
  - ‘max’ Retorna o máximo da lista
  - ‘min’ Retorna o mínimo da lista

- **Comandos úteis para Listas**
  - ‘max’ Retorna o máximo da lista
  - ‘min’ Retorna o mínimo da lista
  - ‘sum’ Retorna a soma dos elementos

CEFET-MG

- **Comandos úteis para Listas**
  - ‘max’ Retorna o máximo da lista
  - ‘min’ Retorna o mínimo da lista
  - ‘sum’ Retorna a soma dos elementos
  - Ex.:

```
>>> print(max(nums))
>>> print(min(nums))
>>> print(sum(nums))
>>> print(sum(nums)/len(nums))
```

- **Exercícios:**

18. Crie uma lista com 20 números inteiro aleatórios entre 1 e 80, e faça:

    a) Qual o maior número?

    b) Qual o menor?

    c) Qual a amplitude dos dados?

    d) Qual a soma?

    e) Qual a média simples?

    f) Qual a média dos 10 primeiros elementos?

    g) Qual o máximo e mínimo dos 10 últimos elementos?

    Dica: use a função randint {>>> from random import randint}

```
import random

# Criando a lista com 20 números inteiros aleatórios entre 1 e 80
numeros = [random.randint(1, 80) for _ in range(20)]
print("Lista de números:", numeros)

# Maior número
maior = max(numeros)
# Menor número
menor = min(numeros)
# Amplitude (diferença entre o maior e o menor)
amplitude = maior - menor
# Soma
soma = sum(numeros)
# Média simples
media = soma / len(numeros)
```

```
# Média dos 10 primeiros elementos
media_10_primeiros = sum(numeros[:10]) / 10
# Máximo e mínimo dos 10 últimos
max_10_ultimos = max(numeros[-10:])
min_10_ultimos = min(numeros[-10:])

# Exibindo os resultados
print("\nResultados:")
print(f"Maior número: {maior}")
print(f"Menor número: {menor}")
print(f"Amplitude dos dados: {amplitude}")
print(f"Soma: {soma}")
print(f"Média simples: {media:.2f}")
print(f"Média dos 10 primeiros: {media_10_primeiros:.2f}")
print(f"Máximo dos 10 últimos: {max_10_ultimos}")
print(f"Mínimo dos 10 últimos: {min_10_ultimos}")
```

CEFET-MG

- **Strings e Listas**
  - .split() Divide uma string e produz uma lista de strings

CEFET-MG

- **Strings e Listas**
  - .split() Divide uma string e produz uma lista de strings
  - Ex.:

```
>>> email = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'

>>> words = email.split()
```

- **Strings e Listas**
  - .split() Divide uma string e produz uma lista de strings
  - Ex.:

```
>>> email = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> words = email.split()
```

  - **Como faço para imprimir somente o nome do email ‘**stephen.marquard’**?**

CEFET-MG

- **Strings e Listas**
  - .split() Divide uma string e produz uma lista de strings
  - Ex.:

```
>>> email = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> words = email.split()
```

  - **Como faço para imprimir somente o nome do email ‘**stephen.marquard’**?**
    - Use o delimitador
    - .split(*delimitador*)

CEFET-MG

- **Strings e Listas**
  - **Como faço para imprimir somente o nome do email ‘**stephen.marquard’**?**

```
>>> email = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> words = email.split()
>>> mail = words[1]
>>> aux = mail.split(‘@’)
>>> nome = aux[0]
```

CEFET-MG

- **Ordenação de uma lista**
  - sort() Ordena de forma permanente.

- **Ordenação de uma lista**
  - sort() Ordena de forma permanente.
  - sorted() Ordena de forma temporária.

- **Ordenação de uma lista**
  - sort() Ordena de forma permanente.
  - sorted() Ordena de forma temporária.
  - Ex.:
    - cars = ['bmw', 'audi', 'toyota', 'subaru']
    - cars.sorted()
    - cars.sort()

- **Ordenação de uma lista**
  - sort() Ordena de forma permanente.
  - sorted() Ordena de forma temporária.
  - Ex.:
    - cars = ['bmw', 'audi', 'toyota', 'subaru']
    - cars.sorted()
    - cars.sort()
  - **E se eu quiser a ordem inversa de ordenação?**

- Ordenação de uma lista
  - E se eu quiser a ordem inversa de ordenação?

- **Ordenação de uma lista**
  - **E se eu quiser a ordem inversa de ordenação?**
  - argumento reverse=True

- **Ordenação de uma lista**
  - **E se eu quiser a ordem inversa de ordenação?**
  - argumento reverse=True
  - Ex.:

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']

>>> cars.sort(reverse=True)
```

- **Ordenação de uma lista**
  - Ordem inversa da lista original
  - .reverse()

CEFET-MG

- **Ordenação de uma lista**
  - Ordem inversa da lista original
  - .reverse()
  - Ex.:

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']

>>> cars.reverse()
```

CEFET-MG

- **Tuplas**
  - Lista imutável
  - Definida pelos parênteses ‘()’
  - Pode ser redefinida

CEFET-MG

- **Tuplas**
  - Lista imutável
  - Definida pelos parênteses ‘()’
  - Pode ser redefinida
  - Ex.:

```
>>> dim_triangulo = (10,40)

>>> dim_triangulo = (15,45)
```

Mãos à obra

- **Exercícios**

19) Pizzas: Pense em pelo menos três tipos de pizzas favoritas. Armazene os nomes dessas pizzas e, então, utilize um laço for para exibir o nome de cada pizza.

a) Modifique seu laço for para mostrar uma frase usando o nome da pizza em vez de exibir apenas o nome dela. Para cada pizza, você deve ter uma linha na saída contendo uma frase simples como Gosto de pizza de pepperoni.

b) Acrescente uma linha no final de seu programa, fora do laço for, que informe quanto você gosta de pizza. A saída deve ser constituída de três ou mais linhas sobre os tipos de pizza que você gosta e de uma frase adicional, por exemplo, Eu realmente adoro pizza!

Programação em Python

CEFET-MG

- # Lista com pelo menos três tipos de pizzas favoritas
- pizzas = ["calabresa", "pepperoni", "frango com catupiry"]

- # Laço for para exibir frases personalizadas
- for pizza in pizzas:
- print(f"Gosto de pizza de {pizza}.")

- # Mensagem final fora do laço
- print("\nEu realmente adoro pizza!")

- **Exercícios**

20) Animais: Pense em pelo menos três animais diferentes que tenham uma característica em comum. Armazene os nomes desses animais em uma lista e, então, utilize um laço for para exibir o nome de cada animal.

a) Modifique seu programa para exibir uma frase sobre cada animal, por exemplo, Um cachorro seria um ótimo animal de estimação.

b) Acrescente uma linha no final de seu programa informando o que esses animais têm em comum. Você poderia exibir uma frase como Qualquer um desses animais seria um ótimo animal de estimação!

Programação em Python

CEFET-MG

```
# Lista com pelo menos três animais que compartilham uma característica
animais = ["cachorro", "gato", "coelho"]


# Laço for para exibir frases personalizadas
for animal in animais:
    print(f"Um {animal} seria um ótimo animal de estimação.")


# Mensagem final fora do laço
print("\nQualquer um desses animais seria um ótimo animal de estimação!")
```

CEFET-MG

- **Exercícios**

21) Contando até vinte: Use um laço for para exibir os números de 1 a 20, incluindo-os.

22) Um milhão: Crie uma lista de números de um a um milhão e, então, use um laço for para exibir os números. (Se a saída estiver demorando demais, interrompa pressionando CTRL-C ou feche a janela de saída.)

23) Somando um milhão: Crie uma lista de números de um a um milhão e, em seguida, use min() e max() para garantir que sua lista realmente começa em um e termina em um milhão. Além disso, utilize a função sum() para ver a rapidez com que Python é capaz de somar um milhão de números.

Programação em Python

CEFET-MG

```
# Exibe os números de 1 a 20
for numero in range(1, 21):
    print(numero)


# Criar uma lista de 1 até 1.000.000
numeros = list(range(1, 1000001))


# Verificar o menor e o maior valor
print("Menor número da lista:", min(numeros))
print("Maior número da lista:", max(numeros))


# Calcular a soma de todos os elementos
print("Soma de todos os números:", sum(numeros))
```

- **Exercícios**

23) Números ímpares: Use o terceiro argumento da função range() para criar uma lista de números ímpares de 1 a 20. Utilize um laço for para exibir todos os números.

24) Três: Crie uma lista de múltiplos de 3, de 3 a 30. Use um laço for para exibir os números de sua lista.

25) Cubos: Um número elevado à terceira potência é chamado de cubo. Por exemplo, o cubo de 2 é escrito como 2**3 em Python. Crie uma lista dos dez primeiros cubos (isto é, o cubo de cada inteiro de 1 a 10), e utilize um laço for para exibir o valor de cada cubo.

Programação em Python

CEFET-MG

```
# Gera números ímpares de 1 a 20
impares = list(range(1, 21, 2))

# Exibe todos os números ímpares
for numero in impares:
    print(numero)

# Gera múltiplos de 3 de 3 até 30
multiplos_de_tres = list(range(3, 31, 3))

# Exibe os múltiplos
for numero in multiplos_de_tres:
    print(numero)

# Cria lista dos 10 primeiros cubos
cubos = [numero**3 for numero in range(1, 11)]

# Exibe os cubos
for cubo in cubos:
    print(cubo)
```

- aqui

Programação em Python

CEFET-MG

- Crie uma lista com os números de 1 a 100.
- Use um laço for para verificar, com condicionais, quais números da lista são primos.
- Armazene os primos em uma nova lista e exiba-a ao final.

- Dica: use operadores lógicos para verificar a divisibilidade.

Programação em Python

CEFET-MG

```
# Lista de 1 a 100
numeros = list(range(1, 101))
primos = []

for n in numeros:
    if n > 1:  # 0 e 1 não são primos
        eh_primo = True
        for i in range(2, int(n**0.5) + 1):
            if n % i == 0:
                eh_primo = False
                break
        if eh_primo:
            primos.append(n)

print("Números primos de 1 a 100:", primos)
```

Programação em Python

CEFET-MG

- Peça ao usuário para digitar uma frase.
- Com um for, percorra a string e:
  - Conte quantas vogais e consoantes existem.
  - Verifique se a frase contém a palavra "engenharia".
  - Inverta a frase e mostre-a na tela.

CEFET-MG

```
frase = input("Digite uma frase: ").lower()
vogais = "aeiou"
qtd_vogais = 0
qtd_consoantes = 0

for char in frase:
    if char.isalpha():  # apenas letras
        if char in vogais:
            qtd_vogais += 1
        else:
            qtd_consoantes += 1

print(f"Vogais: {qtd_vogais}, Consoantes: {qtd_consoantes}")
print("Contém a palavra 'engenharia'? ", "engenharia" in frase)
print("Frase invertida:", frase[::-1])
```

Programação em Python

CEFET-MG

- Peça ao usuário que digite 10 notas (números reais) e armazene em uma lista.
- Usando for e condicionais:
  - Calcule média, maior, menor e desvio em relação à média.
  - Exiba quais alunos tiveram nota acima da média.
  - Indique se todos os alunos ficaram acima de 5 (use operadores lógicos).

Programação em Python

CEFET-MG

```
notas = []
for i in range(10):
    nota = float(input(f"Digite a nota {i+1}: "))
    notas.append(nota)

media = sum(notas) / len(notas)
maior = max(notas)
menor = min(notas)

print(f"Média: {media:.2f}")
print(f"Maior nota: {maior}")
print(f"Menor nota: {menor}")

print("Notas acima da média:")
for i, nota in enumerate(notas):
    if nota > media:
        print(f"Aluno {i+1}: {nota}")

if all(nota > 5 for nota in notas):
    print("Todos os alunos ficaram acima de 5")
else:
    print("Algum aluno ficou com nota 5 ou abaixo")
```

- Solicite ao usuário que crie uma senha (string).
- Verifique com condicionais e laço for:
  - Se possui ao menos 8 caracteres.
  - Se contém ao menos 1 letra maiúscula, 1 minúscula, 1 número e 1 símbolo (@, #, $, %, &, etc.).
  - Se a senha contém espaços, deve ser rejeitada.
  - Mostre mensagens adequadas de aprovação ou falha.

Programação em Python

CEFET-MG

```
senha = input("Crie sua senha: ")

tem_maiuscula = False
tem_minuscula = False
tem_numero = False
tem_simbolo = False
simbolos = "@#$%&*!"

for char in senha:
    if char.isupper():
        tem_maiuscula = True
    elif char.islower():
        tem_minuscula = True
    elif char.isdigit():
        tem_numero = True
    elif char in simbolos:
        tem_simbolo = True

if (len(senha) >= 8 and tem_maiuscula and tem_minuscula
    and tem_numero and tem_simbolo and " " not in
senha):
    print("Senha aceita!")
else:
    print("Senha inválida. Verifique os critérios.")
```

Programação em Python

CEFET-MG

- Crie uma lista com 15 números inteiros aleatórios.
- Use um laço for para somar apenas os números pares.
- Exiba a lista e o resultado.

Programação em Python

CEFET-MG

- numeros = [3, 8, 15, 22, 9, 12, 4, 7, 30, 11, 18, 25, 2, 10, 5]
- soma_pares = 0

- for n in numeros:
- if n % 2 == 0:
- soma_pares += n

- print(numeros)
- print("Soma dos pares:", soma_pares)

Programação em Python

CEFET-MG

- Dada uma tupla de palavras, conte quantas vogais existem em todas as palavras somadas.
- Dica: crie um tupla de vogais

Programação em Python

CEFET-MG

```
palavras = ("engenharia", "computacao", "python", "algoritmos")
vogais = "aeiou"
contador = 0


for palavra in palavras:
    for letra in palavra:
        if letra in vogais:
            contador += 1


print("Quantidade de vogais:", contador)
```

Programação em Python

CEFET-MG

- Dada uma lista de números, exiba apenas os números primos.

Programação em Python

CEFET-MG

```
numeros = [1, 2, 3, 20, 23, 29, 30, 37]

for n in numeros:
    if n > 0:
        primo = True
        for i in range(2, n):
            if n % i == 0:
                primo = False
                break
        if primo:
            print(n, "é primo")
```

Programação em Python

CEFET-MG

- Dada uma lista de números aleatórios, calcule a média.
- Em seguida, exiba os elementos que são **maiores que a média**.

Programação em Python

CEFET-MG

- valores = [10, 15, 8, 20, 30, 25, 5]
- media = sum(valores) / len(valores)

- print("Média:", media)
- print("Maiores que a média:")
- for v in valores:
-     if v > media:
-         print(v)

Programação em Python

CEFET-MG

- Crie uma lista de 10 números aleatórios pares entre 0 e 100.
- Mostre a lista invertida.

Programação em Python

CEFET-MG

- import random

- # Criar lista com 10 números aleatórios pares entre 0 e 100
- numeros = [random.randrange(0, 101, 2) for _ in range(10)]

- print("Lista original:", numeros)

- # Mostrar a lista invertida
- invertida = numeros[::-1]
- print("Lista invertida:", invertida)

Programação em Python

CEFET-MG

- Crie uma lista de listas (matriz) que represente a matriz identidade de ordem n (fornecida pelo usuário).

Programação em Python

CEFET-MG

```
n = int(input("Digite a ordem da matriz identidade: "))
matriz = []

for i in range(n):
    linha = []
    for j in range(n):
        if i == j:
            linha.append(1)
        else:
            linha.append(0)
    matriz.append(linha)

for linha in matriz:
    print(linha)
```

Programação em Python

CEFET-MG

- Dada uma lista com 40 números aleatórios no intervalo (0,100), crie uma nova lista contendo apenas os elementos **únicos** (sem repetição), mantendo a ordem original.

Programação em Python

CEFET-MG

- lista = [random.randint(0, 100) for _ in range(40)]
- unicos = []

- for n in lista:
- if n not in unicos:
- unicos.append(n)

- print("Lista sem repetição:", unicos)

Programação em Python

CEFET-MG

- Dada uma lista de palavras, mostre quais são palíndromos.

Programação em Python

CEFET-MG

- palavras = ["radar", "python", "ana", "computador", "osso"]
- palindromos = []

- for p in palavras:
-     if p == p[::-1]:
-         palindromos.append(p)

- print("Palíndromos:", palindromos)

- Dada uma tupla com três números representando lados de um triângulo, verifique:
  - Se é um triângulo válido
  - Se é equilátero, isósceles ou escaleno.

```
lados = (5, 5, 8)

a, b, c = lados

if a + b > c and a + c > b and b + c > a:
    if a == b == c:
        print("Equilátero")
    elif a == b or b == c or a == c:
        print("Isósceles")
    else:
        print("Escaleno")
else:
    print("Não é um triângulo válido")
```

Programação em Python

CEFET-MG

- Implemente a cifra de César para deslocar cada letra de uma palavra em k posições no alfabeto.
- Considere apenas letras **minúsculas.**

CEFET-MG

```
texto = "engenharia"
k = 3
alfabeto = "abcdefghijklmnopqrstuvwxyz"
criptografado = ""

for letra in texto:
    if letra in alfabeto:
        pos = (alfabeto.index(letra) + k) % 26
        criptografado += alfabeto[pos]
    else:
        criptografado += letra

print("Texto criptografado:", criptografado)
```

Programação em Python

CEFET-MG

- Implemente a cifra de César para deslocar cada letra de uma palavra em k posições no alfabeto.
- Agora considere letras **minúsculas e minúsculas**.

CEFET-MG

```
texto = "engenharia"
k = 3
alfabeto = "abcdefghijklmnopqrstuvwxyz"
criptografado = ""

for letra in texto.lower():
    if letra in alfabeto:
        pos = (alfabeto.index(letra) + k) % 26
        criptografado += alfabeto[pos]
    else:
        criptografado += letra

print("Texto criptografado:", criptografado)
```

Programação em Python

CEFET-MG

- Dado uma palavra cifrada e o parâmetro inteiro, implemente um tradutor da cifra de César para as palavras codificadas no exercício anterior.

CEFET-MG

```
# Entrada do usuário
texto_cifrado = input("Digite o texto cifrado: ")
k = int(input("Digite o valor de k (deslocamento usado na cifra): "))

resultado = ""
alfabeto = "abcdefghijklmnopqrstuvwxyz"

for letra in texto_cifrado.lower():
    if letra in alfabeto:
        pos = (alfabeto.index(letra) - k) % 26
        criptografado += alfabeto[pos]
    else:
        criptografado += letra

print("Texto criptografado:", criptografado)
```

Programação em Python

CEFET-MG

- Escreva uma função que receba duas listas de mesmo tamanho e retorne uma nova lista com:
  - True se o elemento da primeira for maior que o da segunda,
  - False caso contrário.
- Use operadores lógicos e zip().

```
lista1 = [5, 10, 15, 20]
lista2 = [3, 12, 10, 25]

resultado = [a > b for a, b in zip(lista1, lista2)]

print("Lista 1:", lista1)
print("Lista 2:", lista2)
print("Resultado lógico:", resultado)
```

- **DESAFIO: Analisador de Palavras Únicas**
- Crie um programa em Python que leia uma frase digitada pelo usuário.
- Separe todas as palavras dessa frase e armazene-as em uma lista.
- Faça as seguintes operações:
  a) Exiba quantas palavras únicas existem (sem repetições).
  b) Mostre a palavra mais longa e a mais curta da frase.
  c) Crie uma nova lista apenas com as palavras que começam com vogais (a, e, i, o, u).
  d) Inverta cada palavra e exiba uma nova frase formada pelas palavras invertidas.
  e) Some os comprimentos de todas as palavras e exiba a média de caracteres por palavra.
  f) Mostre todas as palavras em ordem alfabética inversa.

- **DESAFIO: Analisador de Palavras Únicas**
- Entrada:
  - Digite uma frase: Programar em Python é muito divertido e poderoso
- Saída esperada:
  - Total de palavras únicas: 8
  - Mais longa: "divertido" | Mais curta: "é"
  - Palavras que começam com vogal: ['em', 'é']
  - Frase invertida: "ramargorP me nohtyP é otium odotrivid e rodorevop"
  - Média de caracteres por palavra: 6.0
  - Palavras em ordem alfabética inversa: ['rodorevop', 'ramargorP', 'otium', 'odotrivid', 'nohtyP', 'me', 'é', 'e']

Programação em Python

CEFET-MG

```
# Ler a frase do usuário
frase = input("Digite uma frase: ")

# Separar em palavras
palavras = frase.split()

# 1) Quantas palavras únicas existem
palavras_unicas = set(palavras)
print("Quantidade de palavras únicas:", len(palavras_unicas))

# 2) Palavra mais longa e mais curta
mais_longa = max(palavras, key=len)
mais_curta = min(palavras, key=len)
print("Palavra mais longa:", mais_longa)
print("Palavra mais curta:", mais_curta)

# 3) Lista com palavras que começam com vogais
vogais = ('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U')
palavras_vogais = [p for p in palavras if p.startswith(vogais)]
print("Palavras que começam com vogais:", palavras_vogais)

# 4) Inverter cada palavra e formar nova frase
invertidas = [p[::-1] for p in palavras]
nova_frase = " ".join(invertidas)
print("Frase com palavras invertidas:", nova_frase)

# 5) Média de caracteres por palavra
tamanhos = [len(p) for p in palavras]
media = sum(tamanhos) / len(palavras)
print("Média de caracteres por palavra:", media)

# 6) Palavras em ordem alfabética inversa
ordenadas_inv = sorted(palavras, reverse=True)
print("Palavras em ordem alfabética inversa:", ordenadas_inv)
```