ESTUDOS PYTHON

- **VARIÁVEIS E ENTRADA DE DADOS**

  Momento em que o programa recebe dados e valores por um dispositivo de entrada de dados.

  - Função ***input***: solicitar dados do usuário

```
nome = input("Digite seu nome: ")
print(f"Olá, {nome}")
```

- Conversão da Entrada de Dados

  A função input é utilizada para entrada de dados do tipo string, para receber dados numéricos, é utilizado dessa forma:

```
#calculando o valor de um bônus por tempo de serviço
anos = int(input("Anos de serviço "))
valor_por_ano = float(input("Valor por ano: "))
bonus = anos * valor_por_ano
print(f"Bônus de R$ {bonus:5.2f}")
```

  # **{bonus:5.2f}:** esse :5.2f significa o tipo da formatação de espaçamento do número do bônus

  O número **5** em `{bonus:5.2f}` indica a **largura mínima do campo**, ou seja, o número mínimo de caracteres que a variável `bonus` ocupará na tela.

  Enquanto `.2`: Define o número de casas decimais. Esse tipo de Formatação se chama **.format**

  Existem outras funções que o **.format** pode oferecer à estrutura do código, como: Alinhamento de texto, Preenchimento de Caracteres, mas o mais comum é a Substituição de Variáveis na estrutura do código do programa.

- **CONDIÇÕES**

  ESTRUTURA DE DECISÃO: testes e caminhos a serem seguidos de acordo com o resultado.

  - *If*: “se”, se a condição for verdadeira, faça algo.
    Ex: um programa que lê dois valores e imprime qual é o maior,
    if a < b
    if b < a
    O resultado verdadeiro será executado e o falso será ignorado. A sequência de execução do programa é alterada de acordo com os valores digitados.

  - *Elif:* substitui um par de **else if** se a primeira condição “if” não for verdadeira, ele testa outra condição, pode haver quantos elif´s precisar. (if: condição 1; elif: condição 2)

- *Else:* usado quando nenhuma das condições anteriores for verdadeira, quando a condição de if for falsa.

** Faça um programa que leia a categoria de um produto e determine o preço pela tabela: / Cat. 1 → R$10,00 /
/ Cat. 2 → R$18,00 /
/ Cat. 3 → R$23,00 /
/ Cat. 4 → R$26,00 /
/ Cat. 5 → R$31,00 /

```
#Categoria x preço, usando estruturas condicionais
categoria = int(input("Digite a categoria do produto: "))
if categoria == 1:
   preço = 10
elif categoria == 2:
   preço = 18
elif categoria == 3:
   preço = 23
else:
   print("Categoria inválida, digite um valor entre 1 e 3!")
   preço = 0
print(f"O preço do produto é: R$ {preço:6.2f}")
```

#JOGO DA ADIVINHAÇÃO UTILIZANDO A ESTRUTURA CONDICIONAL

```
from random import randint
from time import sleep
computador = randint(0,5) # faz o computador randomizar um número
print('-=-' * 10)
print('Vou pensar em um número entre 0 e 5. Tente adivinhar...')
print('-=-' * 10)
jogador = int(input('Em qual número eu pensei? '))#o jogador tenta adivinhar
print('PROCESSANDO...')
sleep(2) #faz o programa esperar um tempo de 2seg para mostrar o resultado
if jogador == computador:
   print('Parabéns, você adivinhou!')
else:
   print("Fracassou, errou o número!")
   print('Pensei no número {} e não no {}'.format(computador, jogador))
```

EXEMPLO → IF DENTRO DO ELSE

```
print('********************************')
print('Bem vindo ao Jogo da Adivinhação!')
print('********************************')

num_secreto = 42
chute_str = input('Digite um número: ')
chute = int(chute_str)

acertou = chute == num_secreto
maior = chute > num_secreto
```

```
menor = chute < num_secreto

if acertou:
   print('Você acertou!')
else:
   if (maior):
       print('Você errou!O seu chute foi maior do que o número secreto.')
   elif (menor):
       print('Você errou!O seu chute foi menor do que o número secreto.')

print('Fim de jogo!')
```

Neste exemplo, podemos ver a condição *if* dentro de outra condição, o *else*. Além disso, ocorreu a formatação visando a melhora da legibilidade do código, implementando variáveis para utilizar nas condições(acertou, maior, menor), excluindo assim a escrita maior que ficaria nas condições (if num secreto == chute)

QUICK QUESTION!

```
acertou = (chute == num_secreto)
```

Qual o tipo dessa variável?

a) bytes
b) int
c) bool
d) str

type: mostra o tipo de determinada variável, mostrando sua classe.

- **REPETIÇÕES**

- executa o mesmo bloco de códigos várias vezes de forma automática.
- também é chamado de ***loops***
- Existem dois principais tipos de loops: **for** e **while.**

- **Loop for(para):** Estrutura de repetição com variável de controle. Usado quando se sabe o número de vezes que o bloco de código irá se repetir, ou seja, quando se sabe qual o limite do loop. Geralmente utilizado também em listas.

- **Loop while(enquanto):** repete um bloco de código enquanto a condição for verdadeira. Diferente do **for**, a estrutura do while pode ou não saber seu limite. Se a condição for verdadeira, ele entra no loop. Se a condição for falsa, o loop é interrompido.
  O loop infinito ocorre quando não há alteração no valor, onde a condição não se torna falsa. Para sair do loop, pode-se utilizar o **break**.
  Break x continue
  break sai do bloco do laço abruptamente, ***continue*** apenas pula para próxima iteração.

Quando se usa *while True* se tem um loop infinito, pois não tem seu ponto de interrupção, pois enquanto a condição for verdadeira, o programa ficará rodando.

Faça um programa que mostre a tabuada de cada valor digitado pelo usuário. O programa tem que ser interrompido quando o número solicitado for negativo.

```
'''nesta estrutura utilizamos o for dentro do while, onde
fica mais fácil fazer a tabuada, pois sabemos que a tabuada
que o número de multiplicadores é de 1 até 10, sendo 10 o
limite'''
while True:
    n = int(input('Quer ver a tabuada de qual valor?: '))
    if n < 0:
        break
    #logo depois de ler o número "input", a gente coloca a
condição de interrupção
    print('-' * 30)
    for c in range(1, 11):# o range ignora o último elemento
        print(f'{n} x {c} = {n*c}')
    print('-' * 30)
print('TABUADA ENCERRADA')
```

- Repetição com while utilizando flags: quando a repetição é “infinita” até uma flag(bandeira) específica, ou seja, o programa roda até um comando específico ser verdadeiro e interromper o loop.
  Por exemplo, quando quero contar vários números, e enquanto eu não digitar “0”, vou ficar contando esses números para sempre. Então, nesse programa, “0” é meu flag, o ponto de parada/interrupção.

```
i = 1

while i < 10:
    print(i)
    i +=1
    #essa linha de cima equivale a i = i + 1
print('Terminou')
print(i)
num = int(input('Digite um número para ver sua tabuada: '))
for c in range(1, 11):
#coloca-se 11 pois se colocasse 10, a tabuada ficaria até o
9, pois nunca conta
#o último elemento
    print('{} x {:2} = {}'.format(num, c, num*c))
```

#algoritmo de tabuada de 1 até 10

```
criancas = ["manu", 'Vini', "Serena"]
for item in criancas:
```

```
   print(item)

canal = "Refatorando"
for letra in canal:
   print(letra)
```

CONTADOR: é uma variável utilizada para contar o números de repetições de ocorrências de um evento; número de repetições do while. Ele interrompe o loop e não deixa que ele seja um loop infinito no código de repetições.

```
fim = int(input("Digite o último número a imprimir: "))
x = 1
while x <= fim:
   print(x)
   x = x + 1
```

X -> É contador. Pois está sendo usado para acompanhar e controlar o número de vezes que o while é executado. Ele inicia com o valor mínimo de 1

Crie um programa que simule as vendas de um livro com o estoque inicial de 5 exemplares.

```
contador = 5
while contador > 0:
    print(f'Venda Realizada! Estoque restante: {contador}')
    contador -= 1

print('Estoque esgotado')
```

#CONTAGEM DE PONTOS DE ACERTOS DE QUESTÕES

```
pontos = 0
questão = 1
while questão <= 3:
   resposta = input(f"Resposta da questão {questão}: ")
   if questão == 1 and resposta == "b":
       pontos = pontos + 1
   if questão == 2 and resposta == "a":
       pontos = pontos + 1
   if questão == 3 and resposta =="d":
       pontos = pontos + 1
   questao = questao + 1
print(f"O aluno fez {pontos} pontos")
```

ACUMULADOR: calcula o total de uma soma em um programa.
Contador: o valor adicionado é constante. Contagem de elementos
Acumulador: o valor adicionado é variável. Soma de elementos

```
n = 1
soma = 0
while n <= 10:
   x = int(input(f"Digite o {n} número: "))
```

```
   soma = soma + x
   n = n + 1
print(f"Soma: {soma}")
```

Nesse programa, é calculada a soma dos números oferecidos pelo usuário de 1 até 10. *(n <=10)* No código acima, a variável “soma” é acumulador e “n” é contador.

x = x + 1 é igual a *x += 1* → Operadores de atribuição especiais.

```
#algoritmo para somar todos os números divisíveis por 3 de 1 até 500
soma = 0 #acumulador
cont = 0 #contador
for c in range(1, 501, 2):
   if c % 3 == 0: #se o número for divisível por 3(resto como resultado 3)
       cont += 1
       soma += c
print(f'A soma de todos os {cont} valores é {soma}')
```

for c in range(1, 501, 2):
1 → valor inicial da sequência; 501 → valor final da sequência; 2 → intervalo dos números(pula de 2 em 2).

Exemplo de utilização do **break** para interromper a repetição do **while.**

```
s = 0
while True:
   v = int(input("Digite um número a somar ou 0 para sair: "))
   if v == 0:
       break
   s += v
print(s)
```

A condição do while é True, ou seja, executará para sempre, pois o valor de true é constante.
F-STRING: permite a substituição do valor de uma variável/expressão dentro de uma string. São mais diretas que o *.format* .

- CORES NO TERMINAL
  \033[<style>;<text_color>;<background_color>m
  - STYLE: 0(none); 1: bold(negrito); 4: underline(sublinhar); 7: negative(inverte as funções)
  - TEXT: cores do texto com códigos de 30 a 37.
  - BACK: cores do background(de fundo). códigos de 40 a 47.

#Exercício utilizando cores para destacar resultados no terminal

```
#Programa que lê um número inteiro e diz se ele é primo ou não
num = int(input('Digite um número: '))
tot = 0
```

```
for c in range(1, num + 1):#sempre fazer isso quando quer analisar tal
número, para eliminar o último elemento
   if num % c == 0:
       print('\033[33m', end='')
       tot += 1
   else:
       print('\033[31m', end='')
#No terminal, vai aparecer destacado de cor amarela (\033m) os números que
podem ser divisíveis do número fornecido
#pelo usuário no input
   print('{} '.format(c), end='')#end é utilizado para deixar os números no
terminal de forma vertical, e se
#quiser separá-los por algo no espaçamento entre eles, é só colocar algo
entre as aspas. Nesse caso, não foi colocado nada
print(f'\n\033[mO número {num} foi divisível {tot} vezes')
if tot == 2:
   print('E por isso ele é PRIMO!')
else:
   print('E por isso ele NÃO É PRIMO!')
```

end=’ ’ → Define o que será adicionado ao final da saída da função `print()`. Por padrão, o valor de `end` é `'\n'` (nova linha). Ou seja, o *end* controla o que aparece no final da linha impressa.

sep=’ ‘ (separador) → Serve para separar os elementos com o que estiver entre as aspas e, nesse caso, quebrar a linha para apresentar no terminal, já que não há nada apresentado entre as aspas. Ou seja, o *sep* controla o separador entre os itens.

EX:
```
dia = 15
mes = 10
ano = 2015
print(dia, mes, ano, sep="/")
15/10/2015
```

DIFERENÇA ENTRE TERMINAL E PROMPT DE COMANDO

- **Terminal:** O terminal interativo do Python é uma ferramenta que permite executar comandos Python diretamente no terminal. É bom para testes rápidos de códigos, porém nele não pode executar arquivos.
- **Prompt de comando:** comando mais geral que permite executar comandos do sistema operacional e outras tarefas relacionadas ao sistema como carregar um arquivo.

- **VARIÁVEIS COMPOSTAS: TUPLAS/LISTAS/DICIONÁRIOS/CONJUNTOS**

- TUPLAS: tipo de dados de variáveis, pode ser visto como uma lista, porém a diferença é que as tuplas são imutáveis, assim não podendo ser alterada após sua criação. São ideais para representação de valores constantes. Os elementos são identificados por índices, também pode usar fatiamento.

- As tuplas são apropriadas quando se deseja garantir que os elementos não sejam alterados após a criação. Isso é útil para dados que devem permanecer constantes ao longo do tempo. Como as tuplas são imutáveis, elas podem ter um desempenho ligeiramente melhor em operações de leitura e acesso a elementos. Isso também as tornam adequadas para situações em que a eficiência é crucial.
- Diferente das listas, as tuplas utilizam os parênteses() para serem identificadas. Pode haver dados de tipos diferentes dentro de uma tupla, como dados de uma pessoa(nome, idade, altura, peso). Tuplas também podem ser criadas a partir de listas, utilizando a função ***tuple.***
  L = [1, 2 , 3]
  T = **tuple**(L) → T = (1, 2, 3)
  Embora não possamos alterar uma tupla depois de sua criação, podemos concatená-las, gerando novas duplas:
  >>> t1 = (1, 2, 3)
  >>> t2 = (4, 5, 6)
  >>> t1 + t2
  (1, 2, 3, 4, 5, 6)

```
lanche = ('Hambumguer', 'Suco', 'pizza', 'pudim')
#FORMAS PARA EXIBIR AS TUPLAS NO TERMINAL
for c in lanche: #método mais comum para exibir elementos na saída(aparece
cada elemento "separadamente")
   print(c)

for cont in range(0, len(lanche)): # cont mostra a posição/ utilizando range
   print(f"Eu vou comer {lanche[cont]} na posição {cont}")

for pos, c in enumerate(lanche): #para mostrar a posição dos elementos
   print(f'Eu vou comer {c} na posição {pos}')
```

- LISTAS: tipo de variável que permite o armazenamento de vários valores, acessados por um índice(que começa a contagem pelo 0). Uma lista pode até conter outra lista(sublistas). O tamanho de uma lista é a quantidade de elementos que ela contém.
- As listas são ideais quando você precisa de uma coleção de elementos que pode ser modificada ao longo do tempo.
  Para adicionar um novo elemento à lista, utiliza-se o método *append*, então o elemento será adicionado ao final da lista. Para adicionar um elemento em uma posição específica da lista, usa-se o método *insert*.
  Para eliminar um elemento de uma lista, utiliza-se o método
  del L[0] →L[0] corresponde à lista e o índice do elemento entre [] que vai ser deletado.
  ou o método *remove(*valor dentro da lista/conteúdo*).* Ambos eliminam o elemento e reposicionam a contagem dos índices.

```
#Cálculo de médias
notas = [6, 7, 5, 8, 9]
```

```
soma = 0
x = 0
while x < 5:
   soma += notas[x]
   x += 1
print(f"Média: {soma / x:5.2f}")
```

A estrutura de repetição foi criada para variar o valor de x, onde ele vai variar enquanto este for menor que 5.
x = 0 → índice do elemento da lista(uma lista de 5 elementos tem índices de 0 a 4; sempre começa com o índice 0).

Uma grande vantagem da utilização da lista é que não precisa criar 5 variáveis para as 5 notas, todas foram armazenadas na lista, utilizando o índice para acessar cada nota/elemento.

```
notas = [0, 0, 0, 0, 0]
soma = 0
x = 0
while x < 5:
   notas[x] = float(input(f"Nota {x}: "))
   soma += notas [x]
   x += 1
x = 0
while x < 5:
   print(f"Nota {x}: {notas[x]:5.2f}")
   x += 1
print(f"Média: {soma / x:5.2f}")
```

Nesse exemplo, podemos dar um valor para as notas, onde o programa solicita a entrada de notas para o usuário na posição **x**. Os colchetes podem representar a posição onde o elemento se encontra na lista.
O primeiro loop coleta as notas e calcula a soma.
O segundo loop exibe cada nota formatada.

- CÓPIA E FATIAMENTO DE LISTAS

```
L = [1 , 2, 3, 4, 5]
V = L[:]
V[0] = 6
print(V)
```

Ao escrever L[:], refere-se a uma nova cópia de L, onde pode alterar V e L de forma independente, pois se referem a áreas diferentes na memória.

O fatiamento ocorre quando queremos obter uma parte específica da lista:

```
lista = [10, 20, 30, 40, 50]
sublista1 = lista[1:4]  # [20, 30, 40]
sublista2 = lista[:3]   # [10, 20, 30]
sublista3 = lista[2:]   # [30, 40, 50]
```

```
sublista4 = lista[-3:]  # [30, 40, 50]
```

*sublista2: o índice final não é selecionado, no caso, ficando apenas até o elemento de índice 2(30). O último valor não entra na contagem.

*sublista4: refere-se ao terceiro elemento a partir do final da lista.

- TAMANHO DE LISTAS
  função **len**(comprimento)**:** retorna o valor do número de caracteres/elementos que tem na lista. Ele também consegue facilitar operações que dependem do tamanho da lista.
  lista = [10, 20, 30, 40]
  tamanho = len(lista) *# Retorna 4*

- Método *replace():* substitui um elemento por outro dentro da Cadeia de elementos no programa.
  ex: frase.replace('Python', 'Android') → onde estaria a palavra python, vai ser substituído por Android.

- Método *sort():* Coloca os elementos da lista em ordem. Se a lista for numérica, ele coloca os números na ordem crescente, se for em letras, ele coloca em ordem alfabética(tanto palavras quando letras separadas).
  Para utilizar a ordem inversa, utiliza-se o valores.sort(*reverse=True)* sendo "valores" a variável.

```
num = [2, 5, 9 ,7]
num.append(1)#adicionou o número 1 ao final da lista
num.sort(reverse=True)
num.insert(2, 0)#inserir na posição 2 o número 0
print(num)
print(f'Esta lista tem {len(num)} elementos')
```

  Terminal: [9, 7, 0, 5, 2, 1]
  Esta lista tem 6 elementos

- Método Upper/Lower/Capitalize: **Upper** deixa os caracteres em maiusculos. Se utilizar em uma frase no programa onde já tenha letra maiuscula, mantém e só as que estiverem em minúsculas são modificadas. **Lower** transforma os caracteres em maiusculos, tendo o efeito contrário ao do Upper. Já o **Capitalize** deixa a cadeia de caracteres numa formatação "formal", apenas a primeira letra de cada palavra da frase fica maiuscula, os demais ficam minúsculos.

```
frase = 'Curso em Vídeo Python'
print(frase.upper().count('O'))
```

  #saída = 3 → pois *.count*('O') está contando quantas vezes a letra "O" aparece.

- Método *strip*(): elimina os espaçamentos inúteis na cadeia de caracteres (espaços no começo e no final, antes do início e depois do final).

```
nome = str(input('Digite seu nome completo: ')).strip()
```

```
#strip já vai eliminar os espaçamentos inúteis que o usuário colocar
print('Analisando seu nome...')
print('Seu nome em maiusculo é {}'.format(nome.upper()))
print('Seu nome em minúsculo é {}'.format(nome.lower()))
print('Seu nome tem ao todo {} letras'.format(len(nome) - nome.count(' ')))
#na contagem de letras do nome, o espaçamento ainda estava sendo contado.
#Por isso, foi colocado o nome.count(' ') para contar os espaçamentos para subtrair
#da contagem das letras, como se fosse a equação (letras) - (espaços)
```

- Split(): gera uma lista com todas as palavras de uma cadeia de caracteres(a partir dos espaçamentos).

```
n = str(input('Digite seu nome completo: ')).strip()
nome = n.split()
#utilizou o split para poder separar cada cadeia de caracteres, ou seja,
#cada nome para uma lista, para poder separar o primeiro e último nome
print("Muito prazer!")
print('Seu primeiro nome é {}'.format(nome[0]))
print('Seu último nome é {}'.format(nome[len(nome)-1]))

#não poderia colocar nome[3] ou nome[4] pois não sabemos quantos sobrenomes o usuário pode solicitar
#por isso, é utilizado o len -1(o primeiro de trás pra frente), para que o último elemento seja conferido
```

- '-'.join(frase): junta todos os elementos de uma lista a partir da utilização de um separador específico.

- Método *append. :* adiciona um elemento à lista durante execução do programa. ex: L.append(valor)

```
>>> L = [ ]
>>> L.append("a")
>>> L
[ 'a' ]
>>> L.append("b")
>>> L
[ 'a', 'b' ]
>>> len(L)        → utilizamos o len para determinar o tamanho da lista L
3
```

```
>>> L = [ ]  → outra forma de adicionar elementos
>>> L += [1]
>>> L
[1]
>>> L += [2]
>>> L
[1, 2]
```

- Método *extend* = adiciona os elementos de uma lista na outra.

```
>>> L = ["a"]
>>> L.append("b")
>>> L
['a', 'b']
>>> L.extend(["c"])
```

>>> L
[‘a’, ‘b’, ‘c’]

Para copiar uma lista dentro de listas:

dados = [Pedro, 25]
pessoas = list()
pessoas.append(dados[:])

ao colocar [:] gera um fatiamento, cópia de dados
Então agora tem uma lista dentro de outra lista, onde dos dados de um estão presentes dentro do outro

Outro exemplo de Listas dentro de listas:
pessoas = [[‘Pedro’, 25], [‘Maria’,19], [‘João’, 31]]
0 1 2 → índices
print(pessoas[0][0])
>>> Pedro
Quando ele pediu [0][0], a saída será de ‘Pedro’, pois ele é o primeiro elemento(índice 0) da primeira lista que está dentro da lista de *pessoas*.
O primeiro [0] é em relação à primeira lista e o segundo [0] é em relação ao primeiro elemento da lista.

```
galera = [['joao', 19], ['ana', 31], ['joaquim', 13], ['maria', 45]]
for p in galera:
   print(f'{p[0]} tem{p[1]} anos de idade.')
```

ALGORITMO PARA CONTAR QUANTOS SÃO MAIORES/MENORES DE IDADE UTILIZANDO LISTAS, CONDICIONAL E REPETIÇÃO

```
galera = list()
dados = list()
totmai = totmen = 0

for c in range(0, 3):
   dados.append(str(input('Nome: ')))
   dados.append(int(input('Idade: ')))
   galera.append(dados[:])
   dados.clear()
for p in galera:
   if p[1] >= 18:
       print(f'{p[0]} é maior de idade.')
       totmai += 1
   else:
       print(f'{p[0]} é menor de idade. ')
       totmen += 1
```

```
print(f'Temos {totmai} maiores e {totmen} menores de idade.')
```

totmai e totmen → contadores que vão adicionando 1 a cada elemento correspondente(se a pessoa é maior/menor de idade) e ao final do loop armazenam o total.

EXERCÍCIO PARA DEFINIR A QUANTIDADE DE NÚMEROS E SUA ORDEM DECRESCENTE

```
valores = []
while True:
   valores.append(int(input('Digite um número: ')))
   resp = str(input('Quer continuar? [S/N] '))
   if resp in 'Nn':
       break
valores.sort(reverse=True)
print(f'Você digitou {len(valores)} valores')
print(f'A ordem decrescente dos valores digitados é de {valores}')
```

- Função *range(intervalo):* gera uma sequência de elementos, onde pode ser útil para loops ou outras operações de iteração, uma vez que ele cria uma lista de elementos que segue uma sequência definida.

```
#para poder fazer uma repetição utilizando for na iteração sem
precisar ficar repetindo
#os prints várias vezes, utilizamos:
for c in range(1, 6):
   print('Oi')
print('FIM')
#neste caso, 'oi' será exibido no terminal 6 vezes
for c in range(1, 6):
   print(c)
print('FIM')
```

Já nesse exemplo, no terminal será exibido 1 até 5(o último elemento não conta)

```
s = 0
for c in range(0, 5):
   n = int(input('Digite um valor: '))
   s += n
print(f'O somatório dos números é de {s}')
```

Neste exemplo, temos um somatório de números que o usuário solicitou utilizando o **for**.

- CRIANDO LISTAS ATRAVÉS DE RANGES:

```
valores = list(range(4,11))
```

valores = 4, 5, 6, 7, 8, 9, 10 (o último elemento é sempre descartado).

03. LISTAS, LAÇOS E EXCEÇÕES

Exercícios

1 - Crie uma lista para cada informação a seguir:

- Lista de números de 1 a 10;
- Lista com quatro nomes;
- Lista com o ano que você nasceu e o ano atual.

2 - Crie uma lista e utilize um loop for para percorrer todos os elementos da lista.

3 - Utilize um loop for para calcular a soma dos números ímpares de 1 a 10.

4 - Utilize um loop for para imprimir os números de 1 a 10 em ordem decrescente.

5 - Solicite ao usuário um número e, em seguida, utilize um loop for para imprimir a tabuada desse número, indo de 1 a 10.

6 - Crie uma lista de números e utilize um loop for para calcular a soma de todos os elementos. Utilize um bloco try-except para lidar com possíveis exceções.

7 - Construa um código que calcule a média dos valores em uma lista. Utilize um bloco try-except para lidar com a divisão por zero, caso a lista esteja vazia.

- **DICIONÁRIOS**
  São estruturas de dados semelhantes às tuplas e listas, mas o diferencial é que neles se pode ter índices literais(personalização de índices).
  O dicionário é composto de um conjunto de chaves e valores. O dicionário consiste em relacionar uma chave a um valor específico.

TUPLAS ( )
LISTAS [ ]
DICIONÁRIOS {}

O dicionário serve como identificador de índices de forma personalizada, diferente de utilizar os números em listas, como por exemplo:

LISTAS:

```
dados = list()
dados.append('Pedro')
dados.append('25')
print(dados[0])
print(dados[1])
```

- **DICIONÁRIOS**

```
dados = dict()
dados = {'nome':'Pedro', 'idade': 25}
print(dados['nome'])
print(dados['idade'])
```

Na estrutura dos dicionários, o *append* não é usado para adicionar elementos. Logo, para adicionar algum elemento ao dicionário basta apenas criar um elemento novo.

```
dados = dict()
dados = {'nome':'Pedro', 'idade': 25}
print(dados['nome'])
print(dados['idade'])
dados['sexo'] = 'M'    '''criou um novo elemento'''
```

Para eliminar algum elemento e seu valor, usa-se o *del*.
Os valores e elementos sempre têm que ter as aspas ao serem identificados e separados por vírgulas.

- VALUES/KEYS/ITEMS
  ** value(): são os valores atribuídos aos elementos do dicionário
  ** keys(): São os "nomes dos elementos"
  ** items(): quando for pra mostrar tanto os valores (value) e os nomes dos elementos(keys)

```
filme = { 'Título': 'Star Wars',
          'Ano': '1977',
          'Diretor': 'George Lucas'
}
print(filme.values())
print(filme.keys())
print(filme.items())
```

Utilizando o laço *for* em dicionários: (vale como se fosse usar o *enumerate*)

```
for k,v in filme.items():
   print(f'O {k} é {v}')
```

>>>O Título é Star Wars
O Ano é 1977 #k = key v = value
O Diretor é George Lucas

```
pessoas = {'nome': 'Gustavo', 'idade': 22, 'sexo': 'M'}
print(f'O {pessoas["nome"]} tem {pessoas["idade"]} anos.')
```

>>>O Gustavo tem 22 anos.
Para fazer o print formatado utilizando dicionários, dentro das chaves precisa colocar as aspas duplas "", uma vez que no print é utilizado as aspas simples.

```
pessoas = {'nome': 'Gustavo', 'idade': 22, 'sexo': 'M'}
for k, v in pessoas.items():
```

```
   print(f'{k} = {v}')
```
>>>nome = Gustavo
    idade = 22
    sexo = M

DICIONÁRIOS DENTRO DE LISTAS

```
brasil = list()
estado1 = {'uf': 'Rio de Janeiro', 'sigla': 'RJ'}
estado2 = {'uf': 'São Paulo', 'sigla': 'SP'}
brasil.append(estado1)
brasil.append(estado2)

print(brasil[0]['uf'])
```
>>>Rio de Janeiro

- Método *.copy():* é possível fazer a cópia de um elemento sem precisar fazer fatiamento, pois no dicionário não aceita o fatiamento [:]

```
estado = dict()
brasil = list()
for c in range(0, 3):
   estado['uf'] = str(input('Unidade Federativa: '))
   estado['sigla'] = str(input('Sigla do Estado: '))
   brasil.append(estado.copy())
for e in brasil:
   for k, v in e.items():
       print(f'O campo {k} tem valor {v}.')
```
>>> O campo uf tem valor Rio de Janeiro.
O campo sigla tem valor RJ.
O campo uf tem valor São Paulo .
O campo sigla tem valor SP.
O campo uf tem valor Maranhão .
O campo sigla tem valor MA.

***range(0, 3)* faz a pergunta dos estados(unid. feder. e sigla 3 vezes)
***brasil.append(estado.copy())* está adicionando as informações fornecidas pelo usuário por meio do input. Faz-se a cópia/fatiamento dos dados pois não está fazendo uma relação entre a lista e o dicionário, apenas quer copiar os dados dele.

EXERCÍCIOS DICIONÁRIOS

1. Faça um programa que leia o nome e média de um aluno, guardando também a situação em um dicionário. No final, mostre o conteúdo da estrutura na tela.
    - Resolução

```
aluno = dict()
aluno ['nome'] = str(input('Nome do aluno: '))
aluno ['média'] = float(input(f'Média de {aluno["nome"]}: '))
if aluno['média'] >= 7:
   aluno['situação'] = 'Aprovado'
elif 5 <= aluno['média'] < 7:
   aluno['situação'] = 'Recuperação'
else:
```

```
   aluno['situação'] = 'Reprovado'
print('-=' * 30)
for k, v in aluno.items():
   print(f'   {k} = {v}')
```

- **FUNÇÕES**
  São especialmente interessantes para isolar uma tarefa específica em um trecho de programa, permitindo que a solução de um problema ou uma linha/função seja reutilizada em outras partes do programa, sem precisar repetir as mesmas linhas. É utilizado para otimização do código.
  Para definirmos uma função, utiliza-se o ***def*** como um novo comando.
  (função sem parâmentro)

```
def lin():
    print('-' * 30)

lin()
print(' CURSO EM VÍDEO ')
lin()
print(' FUNÇÕES EM PYTHON ')
lin()
```

saída:

```
------------------------------
 CURSO EM VÍDEO
------------------------------
 FUNÇÕES EM PYTHON
------------------------------
```

nesse exemplo, utilizou-se a função para não precisar colocar o ——- toda hora

Podemos também armazenar textos dentro das funções para que sejam apresentadas sem ter que colocar o comando *print+frase desejada* o tempo todo. Para isso, utilizamos nesse exemplo a função *título(txt)*, onde o *título* é a função e o *txt* é a mensagem que vai aparecer como parâmetro da função, no caso txt = CURSO EM VÍDEO
As primeiras 4 linhas são como se fossem o modelo da função, e a última linha é a mensagem que queremos implementar para apresentar a mensagem dentro dessa função.
(função com parâmetro)

```
def título(txt):
    print('-' * 30)
    print(txt)
    print('-' * 30)

título(' CURSO EM VÍDEO ')
def soma(a, b):
    print(f'A = {a} e B = {b} ')
    s = a + b
    print(f'A soma A + B = {s}')
```

```
soma(4,5)
soma(8,9)
```

Neste exemplo, declaramos uma função de soma que recebe dois números como parâmetros(a, b) e os imprime na tela.