

PROGRAMADOR

Python



CONTEÚDO DO MÓDULO

4

Estrutura de controle

3

Elementos básicos

2

Comentário

Instalação e configuração do Python

7

Matriz

6

Vetor

5

Estrutura de repetição

Conteúdo Aula 8

Instalação e
configuração do
Python

1

Comentário

2

Elementos
básicos

3

INTRODUÇÃO AO PYTHON



V.Office
Treinamentos

Por que aprender Python?

1. Simples e fácil de aprender

- O que contribui para sua simplicidade:
 - **Fonte livre e aberta**
 - **Linguagem de alto nível**
 - **Interpretada**
 - **Grande comunidade**
- Não é preciso lidar com uma sintaxe complexa
- Para um simples Olá mundo, basta uma única linha
 - Essa simplicidade facilita muito o aprendizado da linguagem para iniciantes

Por que aprender Python?

2. Portátil e extensível

- É suportado pela maioria das plataformas de SOs presentes no mercado hoje (**Windows, Linux, Mac OS, Solaris, Playstation, etc**)
- Permite integrar componentes Java e .NET, além de permitir a chamada de bibliotecas do C e C++

Por que aprender Python?

3. Desenvolvimento web

- Possui vários frameworks para desenvolvimento web:

- **Django**
- **Flaske**
- **Pylons**

- Como essas estruturas são escritas em Python, essa é a principal razão que torna o código muito mais rápido e estável.

Por que aprender Python?

4. Inteligência artificial

- Algumas bibliotecas como **Keras** e **TensorFlow** trazem a funcionalidade de aprendizado de máquina para o mix
- Aprendem sem ser explicitamente programando
- Existem também bibliotecas como OpenCv que ajuda no reconhecimento de imagens

Por que aprender Python?

5. Computação gráfica

- Ele é usado para construir aplicativos GUI e de desktop.
- Ele usa a biblioteca **Tkinter** para fornecer uma maneira rápida e fácil de criar aplicativos.
- Ele também é usado no desenvolvimento de jogos, onde você pode escrever a lógica de usar um módulo **pygame**, que também é executado em dispositivos Android.

Por que aprender Python?

6. Enquadramento de testes

- O Python tem muitas estruturas de teste integrados
- Existem muitas ferramentas e módulos para facilitar as coisas:
 - Selenium
 - Splinter
- Suporta testes com multi-plataforma e cross-browser com frameworks como **PyTest** e **Robot Framework**

Por que aprender Python?

7. Big Data

- Possui algumas bibliotecas como:

- Pydoop
- Dask
- Pyspark



Por que aprender Python?

8. Scripting e automação

- O Python pode ser utilizado como uma linguagem de script:
 - O código é escrito em forma de script e executado
 - A máquina lê e interpreta o código
 - A verificação de erros é feita em tempo de execução
- Após ser validado o código, ele pode ser executado várias vezes, podendo assim ser colocado em um script e automatizar algumas tarefas.

Por que aprender Python?

9. Ciência de dados

- Python é uma linguagem muito utilizada por muitos cientistas de dados.
- Acadêmicos e pesquisadores utilizaram durante anos o MATLAB, mas com o lançamento de mecanismos numéricos do Python começaram a utilizá-lo:
 - Numpy
 - Pandas
- O python lida com dados tabulares, matriciais e estatísticos e pode visualizá-los com bibliotecas populares como:
 - Matplotlib
 - Seaborn

Por que aprender Python?

10. Popularidade

- Nos últimos anos o python teve um aumento em sua popularidade.
- Em 2012 o uso do Python estava abaixo de outras 5 linguagens mais populares.
- Hoje em dia está em terceiro lugar, conforme próximo slide:

Por que aprender Python?

	Jan 2019	Jan 2018	Change	Programming Language	Ratings	Change
1	1	Java		Java	16.904%	+2.69%
2	2	C		C	13.337%	+2.30%
3	4	Python	▼	Python	8.294%	+3.62%
4	3	C++	▼	C++	8.158%	+2.55%
5	7	Visual Basic .NET	▼	Visual Basic .NET	6.459%	+3.20%
6	6	JavaScript		JavaScript	3.302%	-0.16%
7	5	C#	▼	C#	3.284%	-0.47%
8	9	PHP	▼	PHP	2.680%	+0.15%
9	-	SQL	❖	SQL	2.277%	+2.28%
10	16	Objective-C	❖	Objective-C	1.781%	-0.08%
11	18	MATLAB	❖	MATLAB	1.502%	-0.15%
12	8	R	❖	R	1.331%	-1.22%
13	10	Perl	▼	Perl	1.225%	-1.19%
14	15	Assembly language	▼	Assembly language	1.196%	-0.86%
15	12	Swift	▼	Swift	1.187%	-1.19%
16	19	Go	▼	Go	1.115%	-0.45%
17	13	Delphi/Object Pascal	❖	Delphi/Object Pascal	1.100%	-1.28%
18	11	Ruby	❖	Ruby	1.097%	-1.31%
19	20	PL/SQL	◀	PL/SQL	1.074%	-0.35%
20	14	Visual Basic	❖	Visual Basic	1.029%	-1.28%

Instalação e configuração do Python

Abaixo temos o link com a documentação do Python:

<https://docs.python.org/3/using/index.html>

Download do Python:

<https://www.python.org/download/>

Instalação e configuração do Python

Abram o programa IDLE e digitem:

- `2 + 3`
 - Irá apresentar o resultado da soma: 5
- `"Olá pessoal"`
 - Irá apresentar: 'Olá pessoal'
- `nome`
 - Dará um erro, pois dirá que não existe uma variável definida com esse nome.
- Teremos que definir uma variável nome antes.
 - `nome = "Graziela"`
- `nome`
 - Agora sim irá apresentar o valor "Graziela" armazenado na variável nome.
- `print(nome)`
 - Agora sim irá apresentar o valor "Graziela" armazenado na variável nome.

Instalação e configuração do Python

Mas até em então estamos utilizando apenas o python com dados simples, e se eu quiser criar um programa, algo mais complexo?

Vamos criar um arquivo chamado: **primeiro.py** dentro da nossa pasta do **Módulo3**.

Nesse arquivo vamos escrever:

```
nome = input("Entre com o nome: ")  
print(nome)
```

Salve o arquivo.

Instalação e configuração do Python

Precisaremos acessar pela linha de comando (**DOS, CMD, Terminal...)**

Acessar a pasta na qual se encontra nosso arquivo **primeiro.py**.

Após estar na nossa pasta, digitem:

```
python primeiro.py
```

Assim estaremos executando nossos programas python pelo terminal.

Comentários

A utilização de comentários da-se tanto para inserirmos uma informações que não podemos esquecer, ou então, algo que desejamos que outros programadores saibam no momento em que estiverem lendo determinadas linhas de código.

A primeira utilização dos comentários é fazer o que o próprio nome sugere: comentar os nossos códigos informando ao interpretador que essas linhas não devem ser executadas.

Comentários

- Comentando uma única linha:

#Esta linha estará comentada.

- Comentando várias linhas:

Utilizamos 3 aspas simples seguidas ou 3 aspas duplas seguidas:
""

Todas as linhas abaixo estarão comentadas.
Essa linha também.

'''

:::::

Todas as linhas abaixo estarão comentadas.
Essa linha também.

:::::

Comentários

```
nome=input("Digite o nome: ")
print (nome)

#print ("Nova linha")
"""

print ("Teste")
print ("Novo teste")
"""

"""

print ("Teste")
print ("Novo teste")
"""

"""

V.OfficE
Ttreinamentos
```

Elementos básicos

• Variáveis

- Variáveis são locais de memória reservados para armazenar valores. Isso significa que quando você cria uma variável, você reserva um espaço na memória.
- Com base no conteúdo que será armazenado na variável o sistema gera um espaço na memória de tamanho variável.
- A declaração acontece automaticamente quando um valor é atribuído a uma variável. O sinal de igual (=) é usado para atribuir valores a variáveis.
- Cada variável tem um nome exclusivo (identificador).

- **counter = 100** # um número inteiro
- **miles = 1000.0** # um número de ponto flutuante
- **name = "John"** # uma string

Elementos básicos

- **Atribuições múltiplas**

- Python permite atribuir um mesmo valor a várias variáveis ao mesmo tempo.
 - `a = b = c = 1`
 - Também pode ser atribuído vários objetos de uma só vez.
 - `a = b = c = 1, 2, "John"`
- Caso queiram formatar o número de casas decimais de um número real, que será apresentado na tela:
 - `print("{:.2f}".format(1/3))`

Elementos básicos

Identificadores:

Um identificador Python é um nome usado para identificar uma variável, função, classe, módulo ou outro objeto. Um identificador começa com uma letra de “A - Z” ou de “a - z” ou um sublinhado (_) seguido por zero ou mais letras, sublinhados e dígitos (0 a 9).

Elementos básicos

Regras de nomenclatura para identificadores em Python:

- Um identificador não pode iniciar por número ex: 1variable, mas variable1 é correto;
- Python não permite caracteres de pontuação como !, @, #, \$, % etc nos identificadores;
- Python é case sensitive logo as variáveis ManPower e manpower são objetos diferentes.

Exemplo de identificadores válidos:

- myClass
- var_1 Print_this_to_screen
- This_is_a_long_variable

Elementos básicos

Podemos ainda utilizar o estilo de escrita conhecido por camelcase, isto é, capitalizar todas as primeiras letras da palavra.

Ex: camelCaseExample

Elementos básicos

Palavras reservadas:

and	exec	not	as
finally	or	assert	for
pass	break	from	print
class	global	raise	continue
if	return	def	import
try	del	in	while
elif	is	with	else
lambda	yield		

Elementos básicos

Blocos de indentação

Blocos de código são denotados por indentação de linha, que é rigidamente aplicada.

O número de espaços no recuo é variável, mas todas as instruções dentro do bloco devem ser indentadas na mesma quantidade.

```
if true:  
    print("True") #1 tab  
else:  
    print("False") #1 tab
```

Elementos básicos

Operadores lógicos - booleanos:

O tipo de dados booleano pode armazenar 2 valores True ou False.

Usamos os operadores booleanos para fazer comparações e controlar o fluxo do programa.

AND - todas os valores precisarão ser True para que retorne True

OR - Apenas um dos valores basta ser True para que retorne True

NOT - Sempre verificará que o valor é False

Elementos básicos

Operadores de comparação:

Esses operadores comparam os valores em ambos os lados e decidem a relação entre eles. Eles também são chamados de operadores relacionais.

São eles:

- `==` - retorna True se os valores foram iguais, do contrário, False.
- `!=` - retorna True sem os valores não forem iguais, do contrário, False.

Elementos básicos

Operadores de comparação:

- > - retorna True se o valor da esquerda for maior que o valor da direita, do contrário, False.
- < - retorna True se o valor da esquerda for menor que o valor da direita, do contrário, False.
- >= - retorna True se o valor da esquerda for maior ou igual ao valor da direita, do contrário, False.
- <= - retorna True se o valor da esquerda for menor ou igual ao valor da direita, do contrário, False.

Elementos básicos

Operadores de atribuição:

Operadores de atribuição são usados no Python para atribuir valores a variáveis.

```
x=10 #atribui o valor 10 a variável x  
x+=1 #o mesmo que x=x+1  
x-=2 #o mesmo que x=x-2  
x *= 4 #o mesmo que x = x * 4  
x //=2 #Divisão exata, o mesmo que x = x // 2  
x%/=4 #resto da divisão,o mesmo que x=x%4  
x **= 5 #potencia, o mesmo que x = x ** 5  
x /= 8 #divisão real,o mesmo que x = x /8
```

Elementos básicos

Aplicação de dados de cadastro:

```
nome = input("Entre com o nome: ")
idade = int(input("Entre com a idade: "))
salario = float(input("Entre com o salário: "))
isFeminino = bool(int(input("É do sexo feminino? (1 - Sim / 0 -
Não): "))
```

```
print(nome)
print(idade)
print(salario)
print(isFeminino)
```



Exercícios

1. Faça um programa que leia dois números inteiros digitados pelo teclado e imprima a soma deles.
2. Faça um programa que leia dois números reais A e B digitados pelo teclado e imprima a divisão de A por B.
3. Faça um programa que leia 5 notas, calcule a média dessas notas e apresente na tela a média.
4. Faça um programa, que obtenha do usuário o valor do produto e a quantidade deste produto que está sendo comprado e ao fim apresente em tela o valor que deverá ser pago.
5. Faça um programa que leia 2 números do usuário, e após isso retorne se esses 2 números são iguais.



Exercícios

6. Faça um programa que calcule a área de um triângulo, considerando a fórmula $\text{area} <- (\text{base} * \text{altura}) / 2$. Utilize as variáveis area, base e altura (obter esses dados através do leia) e os operadores aritméticos de multiplicação e divisão. Após realizado os cálculos apresentar na tela através do escreva.
7. Faça um programa que leia dois valores para as variáveis A e B e efetue a troca dos valores de forma que a variável A passe a possuir o valor da variável B e a variável B passe a possuir o valor da variável A. Apresentar (escreva) os valores trocados. (Note que para haver a troca antes de serem trocados os valores, um deles deverá ser armazenado em outro local para não perder seu valor.)



Exercícios

8. Calcule a quantidade de litros de combustível gasta em uma viagem, utilizando um automóvel que faz 12 k/l. Para obter o cálculo, o usuário deve fornecer (leia) o tempo gasto na viagem e a velocidade média durante ela. Desta forma, será possível obter a distância percorrida com a fórmula $distancia <- tempo * velocidade$. Tendo o valor da distância, basta calcular a quantidade de litros de combustível utilizada na viagem com a fórmula: $litros <- distancia \text{ div } 12$. O programa deve apresentar (escreva) os valores da velocidade média, tempo gasto na viagem, a distância percorrida e a quantidade de litros utilizada na viagem.

Estrutura de controle

As estruturas de controle são utilizadas para modificar o fluxo linear de um programa. Sem as estruturas de controle o nosso programa iria executar uma instrução após outra sempre no sentido início->fim. Com as estruturas de controle podemos decidir ir por diferentes caminhos, seja realizar um laço localizado, voltar ao início, pulas algumas instruções em função de algum evento que ocorreu, etc...

Estrutura de controle

Palavras chave utilizadas em estruturas de controle no Python

if -> se uma dada condição for verdadeira, executa um trecho de código.

elif serve para quando você está testando mais de uma condição, na forma "se este caminho não for válido, este outro é"?

else -> se for falso executa outro trecho.

Estrutura de controle



Utilizando **if** e **else**, estrutura simples:

```
idade= int(input ("Quantos anos você tem?"))
if idade >= 16:
    print ("Você já tem idade para votar. Escolha bem o melhor para o país.\n")
else:
    print ("Deixe o vídeo game de lado e vá brincar no parque.\n")
```

Utilizando **if**, **elif** e **else**, estrutura condicional aninhada:

```
idade= int(input ("Quantos anos você tem?"))
if idade >= 16:
    print ("Você já tem idade para votar. Escolha bem o melhor para o país.\n")
elif idade > 10 and idade < 16:
    print ("Você já tem idade para dominar o Python.\n")
else:
    print ("Deixe o vídeo game de lado e vá brincar no parque.\n")
```

Estrutura de controle

```
idade = int(input('Digite sua idade: '))
if idade >= 10 and idade < 20:
    print('Você é adolescente')
elif idade >= 20 and idade < 30:
    print('Você é jovem')
elif idade >= 30 and idade <= 100:
    print('Você é adulto')
else:
    print('Valor não encontrado!')
```

Estrutura de controle

Em Python não temos o selecione caso, como nas outras linguagens, por isso utilizamos apenas as estruturas aninhadas com os **if, elif e else**.





Exercícios

1. Faça um programa que leia um número inteiros e informe se esse número é par ou ímpar.
2. Faça um Programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido - Caso o usuário digite algo diferente de "F" ou "M".
3. Faça um programa para a leitura de duas notas parciais de um aluno. O programa deve calcular a média alcançada por aluno e apresentar:
 - a. A mensagem "Aprovado", se a média alcançada for maior ou igual a sete;
 - b. A mensagem "Reprovado", se a média for menor do que sete;
 - c. A mensagem "Aprovado com Distinção", se a média for igual a dez.



Exercícios

4. Faça um programa que pergunte o preço de três produtos e informe qual produto você deve comprar, sabendo que a decisão é sempre pelo mais barato.
 5. Faça um Programa que pergunte em que turno você estuda. Peça para digitar M-matutino ou V-Vespertino ou N- Noturno. Imprima a mensagem "Bom Dia!", "Boa Tarde!" ou "Boa Noite!" ou "Valor Inválido!", conforme o caso.
 6. Faça um Programa que leia um número inteiro menor que 1000 e imprima a quantidade de centenas, dezenas e unidades do mesmo.
- Exemplos:
- $326 = 3$ centenas, 2 dezenas e 6 unidades
- $12 = 1$ dezena e 2 unidades

Exercícios

7. Faça um programa que faça 5 perguntas para uma pessoa sobre um crime. As perguntas são:
- "Telefonou para a vítima?"
"Esteve no local do crime?"
"Mora perto da vítima?"
"Devia para a vítima?"
"Já trabalhou com a vítima?"
- O programa deve no final emitir uma classificação sobre a participação da pessoa no crime.
- Se a pessoa responder positivamente a 2 questões ela deve ser classificada como "Suspeito".
Entre 3 e 4 questões como "Cúmplice".
5 questões como "Suposto assassino".
Caso contrário, ele será classificado como "Inocente".

Estrutura de repetição

Permite que um bloco de comandos seja executado diversas vezes.

Podemos utilizar os seguintes tipos de repetição:

- **Repetição condicional:** executa um bloco de código enquanto uma condição lógica for verdadeira (**while**);
- **Repetição contável:** executa um bloco de código um número predeterminado de vezes (**for**).

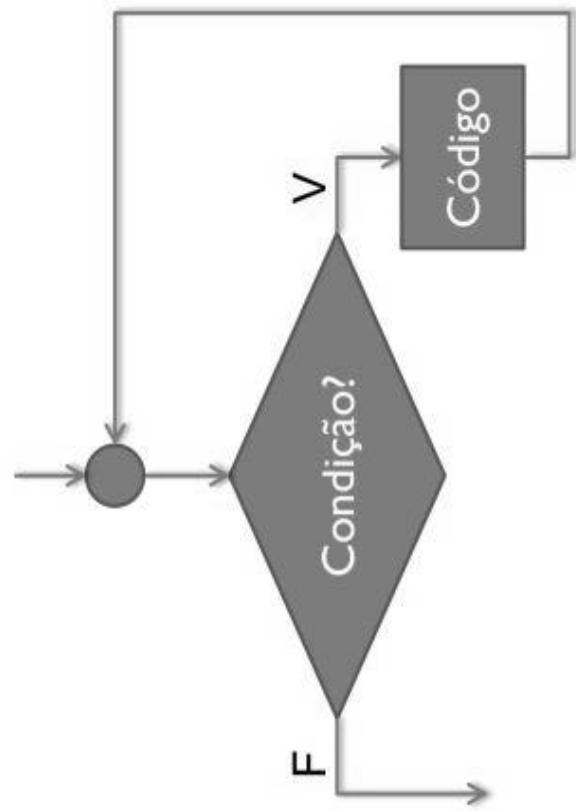
Estrutura de repetição - Condicional

while condição:

instrução 1

instrução 2

instrução 3



Estrutura de repetição - Condicional

Executa o bloco de instruções enquanto a condição for verdadeira.

A condição é uma expressão booleana que pode fazer uso de quaisquer operadores.

O bloco de código pode conter um ou mais comandos.

O início e fim do bloco são definidos de acordo com a indentação.

(Mesmas regras do if ou elif)

A estrutura de repetição é chamada de loop porque continua-se voltando ao início da instrução até que a condição se torne false.

Deve haver algum processo dentro do bloco de comandos que torne a condição false para que a repetição seja encerrada.

Estrutura de repetição - Condicional

Quando a condição se torna falsa, a próxima instrução após o bloco do **while** é executada.

Se a condição do **while** for falsa desde o início, o bloco de instrução nunca é executado.



Estrutura de repetição - Condicional

Exemplo1:

```
numero = int(input("Digite um número: ''))
while numero > 0:
    numero = numero - 1
    print(numero)
print('Boom!!')
```

Estrutura de repetição - Condicional

Exemplo2 - Contador:

Programa que imprime a quantidade de números pares de 100 até 200, incluindo-os.

```
num = 100  
quantidadePares = 0  
while num <= 200:  
    if num % 2 == 0:  
        quantidadePares+=1  
    num+=1  
print(quantidadePares)
```

Estrutura de repetição - Contável

Os valores podem ser listados explicitamente:

```
for x in (0, 1, 2, 3, 4):  
    print(x)
```

O valor de x será repetido 5 vezes, uma para cada elemento informado entre os parênteses.

Estrutura de repetição - Contável

Os valores podem ser especificados como um intervalo com **início**, **fim** e **incremento** usando o **range**.

Exemplo:

```
for x in range(0, 5, 1):
```

```
    print(x)
```

incremento (opcional) – quando omitido, incremento = 1

início (opcional) – quando omitido, início = 0

fim (obrigatório)

Estrutura de repetição - Contável

Podemos colocar apenas `range(6)`, dessa forma estaremos explicitando que irá do 0 até o 5, que terá 6 elementos e irá incrementar de 1 em 1.

Exemplo:

```
for x in range(6):  
    print(x)
```

Estrutura de repetição - Contável

Podemos colocar apenas `range(1, 6)`, dessa forma estamos explicitando que irá do 1 até o 5 (menor do que o número que informar ali) e irá incrementar de 1 em 1.

Exemplo:

```
for x in range(1, 6):  
    print(x)
```

Estrutura de repetição - Contável

Podemos colocar apenas `range(10, 0, -2)`, dessa forma estamos explicitando que irá do 10 até 2 (menor do que o número que informar ali 0) e irá incrementar de -2 em -2.

Exemplo:

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



Exercícios

1. Imprimir a quantidade de números pares entre dois números solicitados para o usuário.
2. Imprimir a soma de todos os números pares entre dois números solicitados para o usuário.
3. Faça um programa que peça uma nota, entre zero e dez. Mostre uma mensagem caso o valor seja inválido e continue pedindo até que o usuário informe um valor válido.
4. Faça um programa que leia um nome de usuário e a sua senha e não aceite a senha igual ao nome do usuário, mostrando uma mensagem de erro e voltando a pedir a senha.
5. Faça um programa que leia 5 números e informe o maior número.



Exercícios

6. Faça um programa que leia 5 números e informe a soma e a média dos números.
7. Faça um programa que peça para n (solicitar a quantidade para o usuário) pessoas a sua idade, ao final o programa deverá verificar a média de idade da turma e então informar conforme a classificação abaixo:
 - 0 e 25 - Jovem
 - 26 e 60 - Adulto
 - Maior que 60 - Idoso

Vetor

Exemplo Motivacional:

Programa para auxiliar a escrever “Parabéns!” nas melhores provas de uma disciplina com 3 alunos:

- Ler os nomes e as notas de 3 alunos;
- Calcular a média da turma;
- Listar os alunos que tiverem nota acima da média;



Vetor

```
nome1 = input('Informe o nome do aluno 1: ')
nome2 = input('Informe o nome do aluno 2: ')
nome3 = input('Informe o nome do aluno 3: ')
nota1 = eval(input('Informe a nota de ' + nome1 + ':'))
nota2 = eval(input('Informe a nota de ' + nome2 + ':'))
nota3 = eval(input('Informe a nota de ' + nome3 + ':'))
media = (nota1 + nota2 + nota3) / 3
print('A média da turma foi: ', media)
if nota1 > media:
    print('Parabéns', nome1)
if nota2 > media:
    print('Parabéns', nome2)
if nota3 > media:
    print('Parabéns', nome3)
```



Vetor

E se fossem 40 alunos?

É possível definir variáveis que guardam mais de um valor de um mesmo tipo;

Essas variáveis são conhecidas como variáveis compostas, variáveis subscriptas, variáveis indexáveis ou arranjos (array)

Em Python existem três tipos principais de variáveis compostas:

- **Listas**
- **Tuplas**
- **Dicionários**

Vetor



Variável composta unidimensional.

- Contém espaço para armazenar diversos valores;
- É acessada via índice;

A ideia de vetor é comum na matemática, com o nome de variável subscrita.

- Exemplo: x_1, x_2, \dots, x_n

O que vimos até agora são variáveis com somente um valor.

- Exemplo: $y = 123$

No caso de vetores, uma mesma variável guarda ao mesmo tempo múltiplos valores.

- Exemplo: $x_1 = 123. x_2 = 456, \dots$
- $x = [123, 456, \dots]$

Lista

- Em outras linguagens de programação, listas são chamadas de vetores e possuem restrições que Python não impõe:**
- Em Python, os valores de uma lista podem ser de qualquer tipo;
 - Em outras linguagens, os valores precisam ser do mesmo tipo;

Em Python:

```
lista = ['A', 1,02, 'Casa', 2.3]
notas = [10, 5, 6.7, 2, 7.5]
```

Utilização de listas

Para acessar (ler ou escrever) uma posição do vetor, basta informar a posição entre colchetes:

```
notas = [8.0, 5.5, 1.5]
media = (notas[0] + notas[1] + notas[2]) / 3
```

notas	0	8.0
	1	5.5
	2	1.5
	media	5.0

Utilização de listas

Pode-se iterar por todos os seus valores usando um comando `for`:

```
notas = [8.0, 5.5, 1.5]
for i in range(3):
    print(notas[i])
```

Criação de uma lista a partir de valores lidos do teclado

Armazenar as notas de 3 alunos em uma lista. A nota de cada aluno será informada pelo teclado:

```
notas[0] = eval(input('Digite a nota do primeiro aluno: '))
notas[1] = eval(input('Digite a nota do segundo aluno: '))
notas[3] = eval(input('Digite a nota do terceiro aluno: '))
```

O que é o **eval**?

Ele transforma o String em Objeto, ou seja em qualquer tipo de dado, não apenas String, ou Inteiro, entre outros.

Criação de uma lista a partir de valores lidos do teclado

Armazenar as notas de 3 alunos em uma lista. A nota de cada aluno será informada pelo teclado:

```
notas[0] = eval(input("Digite a nota do primeiro aluno: ''))
notas[1] = eval(input("Digite a nota do segundo aluno: ''))
notas[3] = eval(input("Digite a nota do terceiro aluno: '))
```

```
Digite a nota do primeiro aluno: 8
Traceback (most recent call last):
  File "/Users/vanessa/workspace/PyCharmProjects/AloMundo/notas.py",
    line 1, in <module>
      notas[0] = eval(input("Digite a nota do primeiro aluno: '"))
NameError: name 'notas' is not defined
```

Process finished with exit code 1

É preciso primeiro criar a lista...

Como não sabemos o que colocar em cada posição da lista, vamos criar uma lista vazia;

```
notas = []
```

Depois vamos adicionar valores na lista usando **append**:

```
n = eval(input('Digite a nota o primeiro aluno: '))
notas.append(n)
```

Voltando ao exemplo

Armazenar as notas de 3 alunos em uma lista. A nota de cada aluno será informada pelo teclado.

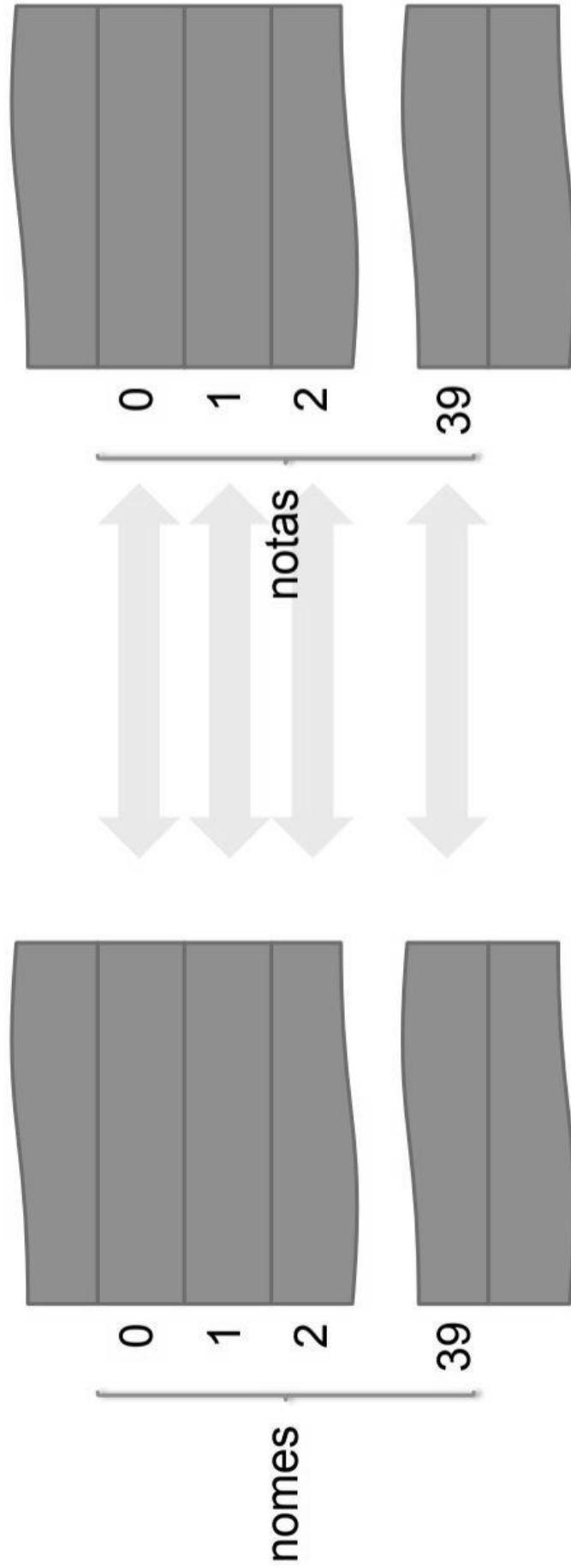
```
notas = []
```

```
notas.append(eval(input('Digite a nota do primeiro aluno: ')))
notas.append(eval(input('Digite a nota do segundo aluno: ')))
notas.append(eval(input('Digite a nota do terceiro aluno: ')))

print(notas)
```

Retomando: E se fossem 40 alunos?

Criaríamos dois vetores (nomes e notas) de 40 posições.
Vincularíamos a posição N do vetor de nomes à posição N do vetor de notas.



Retomando: E se fossem 40 alunos?

```
num_alunos = 40
nomes = []
notas = []
media = 0

for i in range(num_alunos):
    nomes.append(input('Informar o nome do aluno: '))
    #Porque nesse caso deixamos sem
    #o eval? Pois todos os elementos serão String nessa lista, então não precisamos converter para objeto.
    nomes.append(eval(input('Informar a nota de ' + nomes[i] + ': ')))
    media = media + notas[i]
```

```
media = media / num_alunos
print('A média da turma é: ', media)
```

```
for i in range(num_alunos):
    if notas[i] > media:
        print('Parabéns ', nomes[i])
```

Cuidados no uso de listas

Certifique-se de que não esteja querendo acessar posição da lista que não existe.

Exemplo:

```
alunos = ['André', 'Lucas', 'Antônio', 'Maria']
print(alunos[4])
```

Cuidados no uso de listas

Certifique-se de que não esteja querendo acessar posição da lista que não existe.

Exemplo:

```
alunos = ['André', 'Lucas', 'Antônio', 'Maria']
print(alunos[4])
```

```
Traceback (most recent call last):
  File "/Users/vanessa/workspace/PyCharmProjects/AloMundo/notas.py"
line 2, in <module>
  print(alunos[4])
IndexError: list index out of range
```

Process finished with exit code 1

Índices para acesso aos elementos da lista

Python permite acesso à lista em ordem crescente ou decrescente de posição:

Primeira posição é 0

Última posição é -1

```
>>> c = [-45, 6, 0, 72, 1543]
>>> c[3]
72
>>> c[-2]
72
>>> c[0] = c[-5]
True
```

Funções de manipulação de listas

- `len(lista)`
 - Retorna o tamanho da lista

```
numeros = [3, 1, 6, 7, 10, 22, 4]  
len(numeros)
```

Resultados será: 7

Exemplo

Programa que lê uma lista do teclado, soma 1 aos elementos da lista e imprime a lista resultante

```
continua = 'S'  
lista = []  
while (continua == 'S' or continua == 's'):  
    n = eval(input('Digite um número: '))  
    lista.append(n)  
    continua = input('Deseja continuar? (s/n): ')  
  
print(lista)  
for i in range(len(lista)):  
    lista[i] = lista[i] + 1  
print(lista)
```

Concatenação de listas

É possível anexar os valores de uma lista em outra usando o operador "+".

```
>>> lista = [1, 2, 3]
>>> lista + [4]
[1, 2, 3, 4]
>>> lista + [4, 5, 6]
[1, 2, 3, 4, 4, 5, 6]
```

Exemplo

Programa que retorna uma lista com todos os números pares entre 2 e um número n, inclusive.

```
n = eval(input('Digite um número: '))
lista = []
for i in range(2, n+1, 2):
    lista = lista + [i]
print(lista)
```

Exemplo

Programa que retorna uma lista com todos os números pares entre 2 e um número n, inclusive, em ordem reversa:

```
n = eval(input('Digite um número: '))
lista = []
for i in range(2, n+1, 2):
    lista = [i] + lista
print(lista)
```

"Multiplicação" de listas

O operador "*" repete n vezes os elementos que já estão na lista

lista * n equivale a lista + lista + ... + lista (n vezes)

```
>>> lista = [1, 2, 3]
>>> lista * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Inicialização de listas com zero

Em diversas situações onde já sabemos de antemão qual será o tamanho da lista, é útil inicializar a lista com o valor 0. Isso evita que precisemos usar o append para adicionar valores

```
>>> tamанho = 10  
>>> lista = [0] * tamанho  
>>> lista  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Exemplo

```
# inicializa vetor de notas com 0
notas = [0] * 3
soma = 0

# preenche vetor de notas, sem usar append
for i in range(3):
    notas[i] = eval(input("Digite a nota do aluno " + str(i) + ": "))
    soma = soma + notas[i]

print("A média da turma é: ", soma/3)
```



Exercícios

1. Faça um programa que leia dois vetores de 3 posições, que representam forças sobre um ponto no espaço 3D, e escreva a força resultante

Dica: força resultante é obtida pela soma dos valores das coordenadas correspondentes nos dois vetores: $(x_1 + x_2), (y_1 + y_2), (z_1 + z_2)$

2. Faça um programa que preencha por leitura um vetor de 10 posições e obtenha por leitura um número para verificar quantas vezes esse número existe no vetor.
3. Faça um programa que preencha por leitura um vetor de 5 posições, e informe a primeira vez que o valor x (lido do teclado) se encontra no vetor. Caso o valor x não seja encontrado, o programa deve imprimir o valor -1.



Exercícios

4. Um dado é lançado 50 vezes, e o valor correspondente é armazenado em um vetor. Faça um programa que determine o percentual de ocorrências da face 6 do dado dentre esses 50 lançamentos.
5. Faça um programa que leia um vetor **vet** de 20 posições. O programa deve gerar, a partir do vetor lido, um outro vetor **pos** que contenha apenas os valores positivos de **vet**.



Matriz

Variável composta multidimensional

É equivalente a um vetor, contudo permite a utilização de diversas dimensões acessadas via diferentes índices

Pode ser pensada como um vetor onde cada célula é outro vetor, recursivamente

Em diversas situações matrizes são necessárias para correlacionar informações



Matriz

Exemplo motivacional

Assumindo que um aluno é avaliado com cinco notas, seria necessário um vetor de cinco posições para guardar as notas de um aluno.

notas	0	1	2	3	4
	10.0	7.0	9.0	5.5	6.0

Matriz

Contudo, assumindo que uma turma tem três alunos, seria necessária uma matriz bidimensional para guardar as notas de todos os alunos de uma turma.

		notas				
		0	1	2	3	4
alunos	0	5.0	4.5	7.0	5.2	6.1
	1	2.1	6.5	8.0	7.0	6.7
	2	8.6	7.0	9.1	8.7	9.3

```
turma = [[5.0, 4.5, 7.0, 5.2, 6.1], [2.1, 6.5, 8.0, 7.0, 6.7], [8.6, 7.0, 9.1, 8.7, 9.3]]
```

Matriz

Contudo, assumindo que uma turma tem três alunos, seria necessária uma matriz bidimensional para guardar as notas de todos os alunos de uma turma.

		notas				
		0	1	2	3	4
alunos	0	5.0	4.5	7.0	5.2	6.1
	1	2.1	6.5	5.0	7.0	6.7
	2	8.6	7.0	9.1	8.7	9.3

```
turma= [[5.0, 4.5, 7.0, 5.2, 6.1], [2.1, 6.5, 5.0, 7.0, 6.7],  
[8.6, 7.0, 9.1, 8.7, 9.3]]
```

Matriz

Na memória seria algo assim:

	0	5.0
	1	4.5
	2	7.0
	3	5.2
	4	6.1
		0
turma	...	
	0	7.0
	1	9.1
	2	8.7
	3	9.3
	4	



Matriz

Acesso aos valores: [linha][coluna]

Segunda nota do primeiro aluno

```
>>> turma[0][1]
```

4.5

Quinta nota do terceiro aluno

```
>>> turma[2][4]
```

9.3

notas

		notas				
		0	1	2	3	4
alunos	0	5.0	4.5	7.0	5.2	6
	1	2.1	6.5	8.0	7.0	6
	2	8.6	7.0	9.1	8.7	9



Matriz

Calcular a média da turma:

```
turma = [[5.0, 4.5, 7.0, 5.2, 6.1], [2.1, 6.5, 8.0, 7.0, 6.7], [8.6, 7.0,  
9.1, 8.7, 9.3]]  
#calcula a média  
media = 0  
#for para percorrer as linhas  
for linha in range(3):  
    #for para percorrer as colunas  
    for coluna in range(5):  
        media = media + turma[linha][coluna]  
media = media / 15  
print(media)
```

Matriz

Preencher a matriz por leitura:

```
turma = []
for linha in range(3):
    # cria linha vazia
    conteudoLinha = []
    for coluna in range(5):
        #vai adicionando as notas na linha
        conteudoLinha.append(int(input('Digite a nota[' + str(linha) + ', ' + str(coluna) + ']:')))

    #adiciona a linha na matriz turma
    turma.append(conteudoLinha)
```



Exemplo

Programa que cria uma matriz n x m preenchida com zeros:

```
l = int(input('Digite a quantidade de linhas da matriz: '))
c = int(input('Digite a quantidade de colunas da matriz: '))
matriz = []
for linhas in range(l):
    linha = []
    for colunas in range(c):
        linha.append(0)
    matriz.append(linha)
print(matriz)
```

Simplificando o exemplo

Programa que cria uma matriz n x m preenchida com zeros:

```
l = int(input('Digite a quantidade de linhas da matriz: '))
c = int(input('Digite a quantidade de colunas da matriz: '))
matriz = []
for linhas in range(l):
    matriz.append([0] * c)
print(matriz)
```

Imprimir em forma de matriz

Programa que cria uma matriz $l \times c$ preenchida com zeros e a imprime no formato de matriz:

```
l = int(input('Digite a quantidade de linhas da matriz: '))
c = int(input('Digite a quantidade de colunas da matriz: '))
matriz = []
for linhas in range(l):
    matriz.append([0]*c)
```

```
#imprimir em formato de matriz
for i in range(l):
    print(matriz[i])
```

Imprimir em formato de matriz

Programa que cria uma matriz $l \times c$ preenchida com zeros e a imprime no formato de matriz:

```
l = int(input('Digite a quantidade de linhas da matriz: '))
c = int(input('Digite a quantidade de colunas da matriz: '))
matriz = []
for linhas in range(l):
    matriz.append([0]*c)
```

```
#imprimir em formato de matriz
for i in range(l):
    print(matriz[i])
```

Resultado para matriz 2

```
[0, 0, 0]
[0, 0, 0]
```

Exemplo Contar Pares

Programa que lê uma matriz 3x3 digitada pelo usuário e conta quantos números pares existem na matriz, imprimindo na tela o resultado e a matriz:

```
matriz = []
for i in range(3):
    linha = []
    for j in range(3):
        linha.append(int(input('Digite o valor de ['+ str(i) + ',' + str(j) + ']:')))
    matriz.append(linha)

#contar pares
pares = 0
for i in range(3):
    for j in range(3):
        if matriz[i][j] % 2 == 0:
            pares = pares + 1

#printar em formato de matriz
for i in range(3):
    print(matriz[i])
#printar qtde de números pares
print('A matriz contém', pares, 'números pares')
```

Relembrando: for iterando sobre valores

Um comando `for` também pode iterar sobre valores de uma

lista:

```
lista = [1,2,4,5,7,8,9]
for i in lista:
    print(i)
```

Variação do Exemplo Contar Pares

```
matriz = []
for i in range(3):
    linha = []
    for j in range(3):
        linha.append(int(input('Digite o valor de [ ' + str(i) + ',' + str(j) + ']:')))
    matriz.append(linha)

#contar pares
pares = 0
for linha in matriz:
    for valor in linha:
        if valor % 2 == 0:
            pares = pares + 1

#printir em formato de matriz
for i in range(3):
    print(matriz[i])
#printir qtde de números pares
print('A matriz contém', pares, 'números pares')
```

Dimensões da matriz

- É possível usar len() para saber a dimensão de uma matriz
- Assumindo que todas as linhas possuem o mesmo número de elementos, pode-se fazer:
 - Número de linhas: len(matriz)
 - Número de colunas: len(matriz[0])

```
matriz = []
parar = False
while not(parar):
    linha = [0] * 10
    matriz.append(linha)
    x = input("Deseja parar? (S/N)")
    if x == "S":
        parar = True
print("A matriz possui", len(matriz), "linhas")
print("A matriz possui", len(matriz[0]), "colunas")
```

Python permite misturar tipos em uma matriz

Exemplo: programa que armazena os nomes e idades de 10 pessoas em uma matriz, e imprime o nome da pessoa mais nova:

Ana	10
Lucas	15
Bia	13
Larissa	24
Leo	21
Bruno	32
Cássio	4
Jonas	8
Lauro	23
Mateus	18

Encontra a pessoa mais nova



```
m = []
#preenche a matriz
for l in range(10):
    linha = []
    linha.append(input('Digite o nome da pessoa ' + str(l) + ':'))
    linha.append(int(input('Digite a idade de ' + linha[0] + ':')))
    m.append(linha)

#procura a pessoa mais nova
menor = m[0][1]
pos = 0
for l in range(10):
    if m[l][1] < menor:
        menor = m[l][1]
        pos = l

#printa a matriz
for l in range(10):
    print(m[l])

print('A pessoa mais nova é', m[pos][0])
```



Exercícios

1. Faça um programa que leia duas matrizes A e B 2×2 de inteiros e imprima a matriz C que é a soma das matrizes A e B.
2. Faça um programa que leia as dimensões de duas matrizes A e B, e depois leia as duas matrizes (sómente deve aceitar números inteiros positivos).
3. Faça um programa que leia uma matriz 3×3 de inteiros e multiplique os elementos da diagonal principal (No qual os índices são iguais, linha e coluna possuem o mesmo valor) da matriz por um número k (obter do usuário). Imprima a matriz na tela antes e depois da multiplicação.



Exercícios

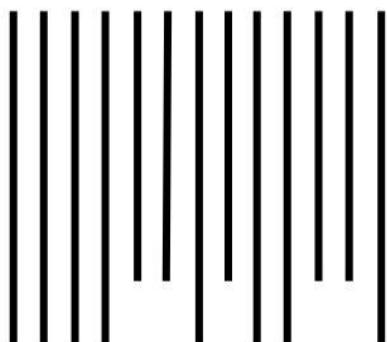
4. Faça um programa que leia uma matriz 3×3 de inteiros e retorne a linha de maior soma. Imprima na tela a matriz, a linha de maior soma e a soma.

5. Faça um programa que leia uma matriz 6×3 com números reais, calcule e mostre:
 - a. o maior elemento da matriz e sua respectiva posição (linha e coluna);
 - b. o menor elemento da matriz e sua respectiva posição.

Programação estruturada ou modular

O que vimos até agora

- Programas usam apenas sequência, repetição e decisão
- Capacidade de resolver diversos problemas, mas difícil de resolver problemas grandes
- Em diversas situações, é necessário repetir o mesmo trecho de código em diversos pontos do programa



Exemplo 1

```
a = [1, 2, 3, 4, 5]
soma = 0
for i in range(len(a)):
    soma = soma + a[i]
media = soma/len(a)
print(media)
```

```
b = [10, 20, 30, 40]
soma = 0
for i in range(len(b)):
    soma = soma + b[i]
media = soma/len(b)
print(media)
```

Exemplo 1

```
a = [1, 2, 3, 4, 5]
soma = 0
for i in range(len(a)):
    soma = soma + a[i]
media = soma/len(a)
print(media)
```



```
b = [10, 20, 30, 40]
soma = 0
for i in range(len(b)):
    soma = soma + b[i]
media = soma/len(b)
print(media)
```

Trecho se
repete 2
vezes

Exemplo 2

1. Ler vetor um vetor A de 10 posições de inteiros
2. Ordenar o vetor A
3. Ler um vetor B de 10 posições de inteiros
4. Ordenar o vetor B
5. Multiplicar o vetor A pelo vetor B, e colocar o resultado num vetor C
6. Ordenar o vetor C

Operação de ordenação do vetor é repetida 3 vezes

Problemas desta “repetição”

Programa muito grande, porque tem várias “partes repetidas”
Erros ficam difíceis de corrigir (e se eu esquecer de corrigir o
erro em uma das N repetições daquele trecho de código?)

O custo da correção do defeito



Solução: subprogramação

Definir o trecho de código que se repete como uma "função" que é chamada no programa.

A função é definida uma única vez, e chamada várias vezes dentro do programa.



Voltando ao exemplo 1

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media
a = [1, 2, 3, 4, 5]
print(calcula_media(a))

b = [10, 20, 30, 40]
print(calcula_media(b))
```

Voltando ao exemplo

```
def calcula_media(v) :  
    soma = 0  
    for i in range(len(v)) :  
        soma = soma + v[i]  
    media = soma/len(v)  
    return media  
  
a = [1, 2, 3, 4, 5]  
print(calcula_media(a))  
b = [10, 20, 30, 40]  
print(calcula_media(b))
```

The diagram illustrates the flow of control through the code. It starts with a large downward-pointing arrow labeled "Definição da função" pointing to the first line of the function definition. This is followed by two smaller arrows pointing to the assignment statements within the loop, both labeled "Chamada da função". Finally, there are two more arrows pointing to the print statements at the bottom, also labeled "Chamada da função".

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma/len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

↓

Execução começa no primeiro comando que está **fora de uma função**

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

Fluxo de Execução

```
def calcula_media(v):
    soma = 0
    for i in range(len(v)):
        soma = soma + v[i]
    media = soma / len(v)
    return media

a = [1, 2, 3, 4, 5]
print(calcula_media(a))
b = [10, 20, 30, 40]
print(calcula_media(b))
```

Declaração de Função

```
def nome_funcao (parametro1, parametro2, parametro3, ...,  
parametron):  
<comandos>  
[return <variável ou valor>]
```

Exemplo:

```
def calcula_media(v):  
    soma = 0  
    for i in range(len(v)):  
        soma = soma + v[i]  
    media = soma / len(v)  
    return media
```

Exemplo

```
def calcula_tempo(velocidade, distancia):  
    tempo = distancia/velocidade  
    return tempo  
  
def calcula_distancia(velocidade, tempo):  
    distancia = velocidade * tempo  
    return distancia  
  
t = calcula_tempo(10, 5)  
print(t)  
d = calcula_distancia(5, 4)  
print(d)
```

Lembrem-se

- Um programa Python pode ter 0 ou mais definições de função.
- Uma função pode ser chamada 0 ou mais vezes.
- Uma função só é executada quando é chamada.
- Duas chamadas de uma mesma função usando valores diferentes para os parâmetros da função podem produzir resultados diferentes.

Escopo de Variáveis

- **Variáveis podem ser locais ou globais**

- **Variáveis locais**

- Declaradas dentro de uma função
- São visíveis somente dentro da função onde foram declaradas
- São destruídas ao término da execução da função

- **Variáveis globais**

- Declaradas fora de todas as funções
- São visíveis por TODAS as funções do programa

Exemplo: variáveis locais

```
def calcula_tempo(velocidade, distancia):
    tempo = distancia/velocidade
    return tempo

def calcula_distancia(velocidade, tempo):
    distancia = velocidade * tempo
    return distancia

t = calcula_tempo(10, 5)
print(t)
d = calcula_distancia(5, 4)
print(d)
```



Exemplo: parâmetros também se comportam como variáveis locais

def calcula_tempo(velocidade, distancia):

tempo = distância/velocidade

return tempo

def calcula_distancia(velocidade, tempo):

distancia = velocidade * tempo

return distancia

t = calcula_tempo(10, 5)

print(t)

d = calcula_distancia(5, 4)

print(d)

Exemplo: variáveis globais

```
def calcula_tempo(velocidade, distancia):
    tempo = distancia/velocidade
    return tempo

def calcula_distancia(velocidade, tempo):
    distancia = velocidade * tempo
    return distancia

t = calcula_tempo(10, 5)
print(t)
d = calcula_distancia(5, 4)
print(d)
```

Uso de Variáveis Globais x Variáveis Locais

- **Cuidado com variáveis globais**
 - Dificultam o entendimento do programa
 - Dificultam a correção de erros no programa
 - Se a variável pode ser usada por qualquer função do programa, encontrar um erro envolvendo o valor desta variável pode ser muito complexo
- **Recomendação**
 - Sempre que possível, usar variáveis LOCAIS e passar os valores necessários para a função como parâmetro

Escopo de Variáveis

```
def calcula_tempo(velocidade, distância):
    tempo = distância / velocidade
    return tempo
    velocidade distância tempo

def calcula_distancia(velocidade, tempo):
    distância = velocidade * tempo
    return distância
    velocidade tempo distância

v = 10
t = calcula_tempo(v, 5)
print(t)
d = calcula_distancia(v, t)
print(d)
```

Parâmetros

Quando uma função é chamada, é necessário fornecer um valor para cada um de seus parâmetros

Isso por ser feito informando o valor diretamente

```
t = calcula_tempo(1, 2)
```

ou; Usando o valor de uma variável

```
t = calcula_tempo(v, d)
```

Passagem de Parâmetro

```
def calcula_tempo(velocidade, distancia):
    tempo = distancia / velocidade
    return tempo
```

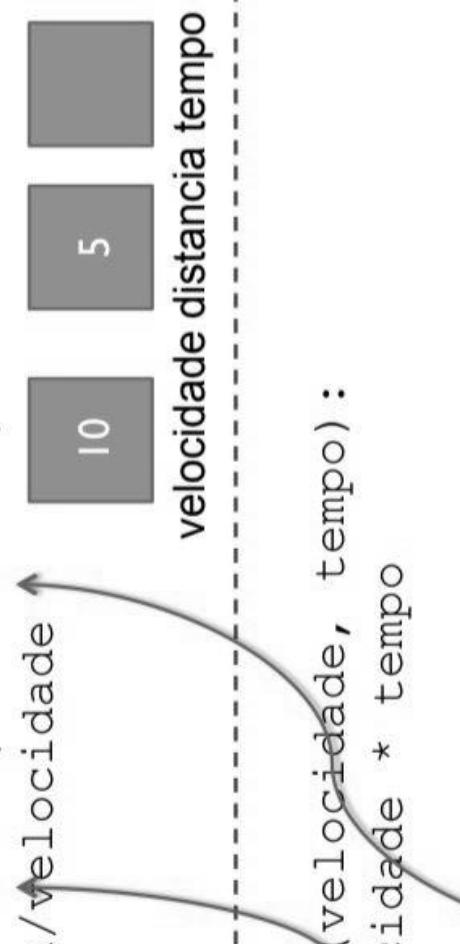
```
def calcula_distancia(velocidade, tempo):
    distancia = velocidade * tempo
    return distancia
```

```
v = 10
t = calcula_tempo(v, 5)
print(t)
d = calcula_distancia(v, t)
print(d)
```



Passagem de Parâmetro

```
| def calcula_tempo(velocidade, distância):  
|     tempo = distância / velocidade  
|     return tempo  
  
| def calcula_distancia(velocidade, tempo):  
|     distância = velocidade * tempo  
|     return distância  
  
v = 10  
t = calcula_tempo(v, 5)  
print(t)  
d = calcula_distancia(v, t)  
print(d)
```



The diagram illustrates the flow of parameters between functions. It shows two calls to `calcula_tempo`. In the first call, the parameter `tempo` receives the value 5 from the argument 5. This value is then returned and assigned to the variable `t` in the second call to `calcula_distancia`. The parameter `tempo` in the second call receives the value 5 from the variable `t`. The parameter `distancia` in the second call receives the value 50 from the expression `velocidade * tempo`.

Passagem de Parâmetro

```
def calcula_tempo(velocidade, distância):
    tempo = distância / velocidade
    return tempo

def calcula_distancia(velocidade, tempo):
    distância = velocidade * tempo
    return distância

v = 10
t = calcula_tempo(v, 5)
print(t)
d = calcula_distancia(v, t)
print(d)
```

Diagram illustrating parameter passing by value:

- Variable **v** (value 10) is passed to the function **calcula_tempo**.
- The function **calcula_tempo** receives a copy of the variable **v** (value 10) and a new variable **tempo** (value 5).
- Inside the function, a new variable **tempo** is created and assigned the value 0.5.
- The function returns the value 0.5.
- The original variable **v** remains unchanged (value 10).

Passagem de Parâmetro por Valor

O valor da variável usada na chamada é copiado para a variável que representa o parâmetro na função

Alterações no valor parâmetro não são refletidas na variável correspondente àquele parâmetro no programa principal



Exemplo

```
def calcula_tempo(velocidade, distancia):
    tempo = distancia/velocidade
    velocidade = 0
    return tempo

def calcula_distancia(velocidade, tempo):
    distancia = velocidade * tempo
    return distancia

v = 10
t = calcula_tempo(v, 5)
print(v)
print(t)
d = calcula_distancia(v, t)
print(d)
```

O valor impresso
por **print(v)**
será **10** ou **0**?

Passagem de Parâmetro por Valor

Python usa passagem de parâmetro por valor

Faz cópia do valor da variável original para o parâmetro da função

Variável original fica preservada das alterações feitas dentro da função

Exceção: **vetores** (ou objetos) funcionam de forma diferente, pois o que é copiado é o **endereço** do vetor, e portanto qualquer alteração é refletida no programa principal -> passagem de parâmetro por referência

Exemplo

```
def maior(vetor):
    vetor.sort()
    return vetor[len(vetor)-1]
v = [5, 4, 3, 2, 1]
print(v)
m = maior(v)
print(m)
print(v)
```

O que será
impresso na tela?

Tipos de passagem de Parâmetro

- Por valor:** o valor da variável na chamada é copiado para a variável da função.
Alterações não são refletidas na variável original

Por referência: é como se o mesmo “escaninho” fosse usado.

Alterações são refletidas na variável original

Retorno das funções

Função que retorna um valor deve usar **return**

Assim que o comando `return` é executado, a função termina

Uma função pode não retornar nenhum valor

Nesse caso, basta **não usar o comando return**

Nesse caso a função termina quando sua última linha de código for executada

Exemplo de função sem retorno

```
def imprime_asterisco(qtd):
    for i in range(qtd):
        print('*****')
        
imprime_asterisco(2)
print('PROGRAMAR EH LEGAL')
imprime_asterisco(2)
```



Chamada de função

Chamada de função

Se a função retorna um valor, pode-se atribuir seu resultado a uma variável

```
m = maior(v)
```

Se a função não retorna um valor (não tem **return**), não se deve atribuir seu resultado a uma variável (se for feito, variável ficará com valor **None**)

```
imprime_asterisco(3)
```

Função sem parâmetro

- Nem toda função precisa ter parâmetro
- Nesse caso, ao definir a função, deve-se abrir e fechar parênteses, sem informar nenhum parâmetro
- O mesmo deve acontecer na chamada da função



Exemplo

```
def menu():
    print('*****')
    print('1 - Somar')
    print('2 - Subtrair')
    print('3 - Multiplicar')
    print('4 - Dividir')
    print('*****')
menu()
opcao = eval(input("Digite a opção desejada: "))
```

Parâmetros default

O valor da variável usada na chamada é copiado para a variável que representa o parâmetro na função

Alterações no valor parâmetro não são refletidas na variável correspondente àquele parâmetro no programa principal



Parâmetros default

- Em alguns casos, pode-se definir um valor default para um parâmetro. Caso ele não seja passado na chamada, o valor default será assumido.
- Exemplo: uma função para calcular a gorjeta de uma conta tem como parâmetros o valor da conta e o percentual da gorjeta. No entanto, na grande maioria dos restaurantes, a gorjeta é sempre 10%. Podemos então colocar 10% como valor default para o parâmetro `percentual_gorjeta`

Exemplo da gorjeta

```
def calcular_gorjeta(valor, percentual=10):  
    return valor * percentual/100
```

```
gorjeta = calcular_gorjeta(400)  
print('O valor da gorjeta de 10% de uma conta de R$ 400 eh',  
     gorjeta)  
gorjeta = calcular_gorjeta(400, 5)  
print('O valor da gorjeta de 5% de uma conta de R$ 400 eh',  
     gorjeta)
```

Quando a gorjeta não é informada na chamada da função, o valor do parâmetro gorjeta fica sendo 10

Uso de Variáveis Globais

- Variáveis globais podem ser acessadas dentro de uma função
- Se for necessário altera-las, é necessário declarar essa intenção escrevendo, no início da função, o comando
global <nome da variável>

Exemplo: variáveis globais acessadas na função

```
def maior():
    if a > b:
        return a
    else:
        return b

a = 1
b = 2
m = maior()
print(m)
```

Péssima prática
de programação!

Exemplo: variável global modificada na função

```
def maior():
    global m
    if a > b:
        m = a
    else:
        m = b
m = 0
a = 1
b = 2
maior()
print(m)
```

Péssima, péssima,
péssima prática
de programação!

Sem uso de variáveis globais: muito mais elegante!

```
def maior(a, b):
    if a > b:
        return a
    else:
        return b

a = 1
b = 2
m = maior(a, b)
print(m)
```

Vejam que agora a e b
são parâmetros.
Os parâmetros também
poderiam ter **outros**
nomes (exemplo, x e y)

Colocar funções em arquivo separado

- Em alguns casos, pode ser necessário colocar todas as funções em um arquivo separado
- Nesse caso, basta definir todas as funções num arquivo .py (por exemplo funcoes.py).
 - Quando precisar usar as funções em um determinado programa, basta fazer **import <nome do arquivo que contém as funções>**
 - Ao chamar a função, colocar o nome do arquivo na frente

Exemplo

Arquivo utilidades.py

```
def soma(v):
    soma = 0
    for i in range(len(v)):
        soma += v[i]
    return soma

def media(v):
    return soma(v)/len(v)
```

Exemplo

Arquivo teste.py

import utilidades

```
v = [1, 3, 5, 7, 9]
print(utilidades.soma(v))
print(utilidades.media(v))
```

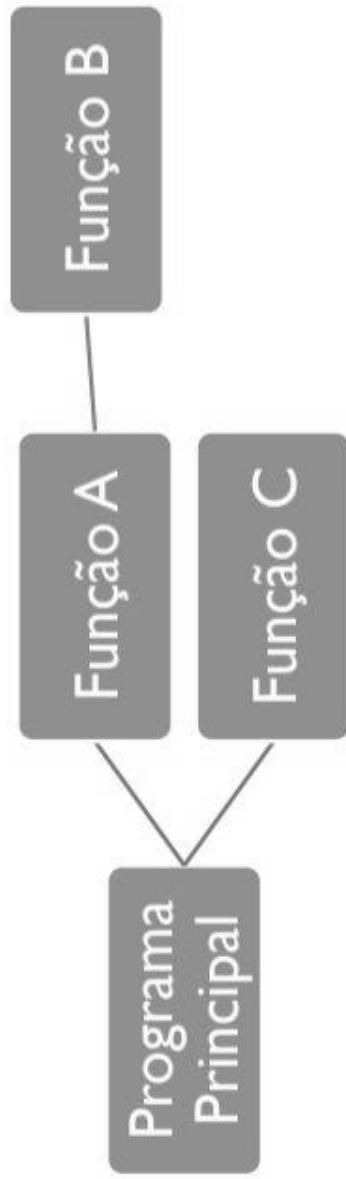


Dividir para conquistar

Antes: um programa gigante



Depois: vários programas menores



Vantagens

- **Economia de código**
 - Quanto mais repetição, mais economia
- **Facilidade na correção de defeitos**
 - Corrigir o defeito em um único local
- **Legibilidade do código**
 - Podemos dar nomes mais intuitivos a blocos de código
 - É como se criássemos nossos próprios comandos
- **Melhor tratamento de complexidade**
 - Estratégia de “dividir para conquistar” nos permite lidar melhor com a complexidade de programas grandes
 - Abordagem top-down ajuda a pensar!



Exercícios

1. Faça um programa, com uma função que necessite de três argumentos, e que forneça a soma desses três argumentos.
2. Faça uma função que informe o status do aluno a partir da sua média de acordo com a tabela a seguir:
Nota acima de 6 à "Aprovado"
Nota entre 4 e 6 à "Verificação Suplementar"
Nota abaixo de 4 à "Reprovado"
3. Faça uma função que receba um número por parâmetro. A função deverá retornar 'P', se seu número for positivo, e 'N', se o número for zero ou negativo.



Exercícios

4. Faça um programa com uma função chamada `somalmposto`. A função possui dois parâmetros formais: `taxalmposto` (que é a quantia de imposto sobre vendas expressa em porcentagem) e `custo` (que é o custo de um item antes do imposto). A função "altera" o valor de custo para incluir o imposto sobre vendas e retorna o mesmo.
5. Faça uma calculadora que obtenha do usuário 2 número e uma das seguintes opções:
 - 1 - Somar
 - 2 - Subtrair
 - 3 - Multiplicar
 - 4 - Dividir
 - 5 - Sair do programa