

Desenvolvimento Desktop

Algoritmo, lógica e
linguagem de programação Python



Sumário

Introdução

Exemplos de Algoritmo

Exemplos de Fluxograma

Linguagens de programação

Linguagem de Programação Python

Links úteis e Bibliografia

Instalação do Python

Instalação do Visual Studio Code

Editor de código e IDE

O que é algoritmo?

Em matemática e ciência da computação, um algoritmo é uma sequência finita de ações executáveis que visam obter uma solução para um determinado tipo de problema. O conceito de algoritmo é frequentemente ilustrado pelo exemplo de uma receita culinária, embora muitos algoritmos sejam mais complexos.

O que são fluxogramas?

Um fluxograma é um diagrama que descreve um processo, sistema ou algoritmo de computador. São amplamente utilizados em várias áreas para documentar, estudar, planejar, melhorar e comunicar processos complexos por meio de diagramas claros e fáceis de entender.



O que é linguagem de programação?

Uma linguagem de programação é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador. Permite que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias.

Exemplo de Algoritmo

Algoritmo Trocar uma lâmpada queimada com o uso de estruturas de repetição

- pegar uma escada
- posicionar a escada debaixo da lâmpada
- buscar uma lâmpada nova
- subir na escada
- retirar a lâmpada queimada
- colocar lâmpada nova
- enquanto lâmpada nova não acender, faça:
 - retirar a lâmpada queimada
 - colocar lâmpada nova

Algoritmo Evolutivo

Gera populacao inicial(Pop)

nº de gerações = 1

melhor individuo = melhor individuo(Pop)

Enquanto nº de gerações < nº máximo de gerações **faça**

Pop₁ = Ø

reproducao(Pop, Pop₁)

Pop = Pop₁

Busca local(Pop)

Atualizar melhor individuo

nº de gerações++

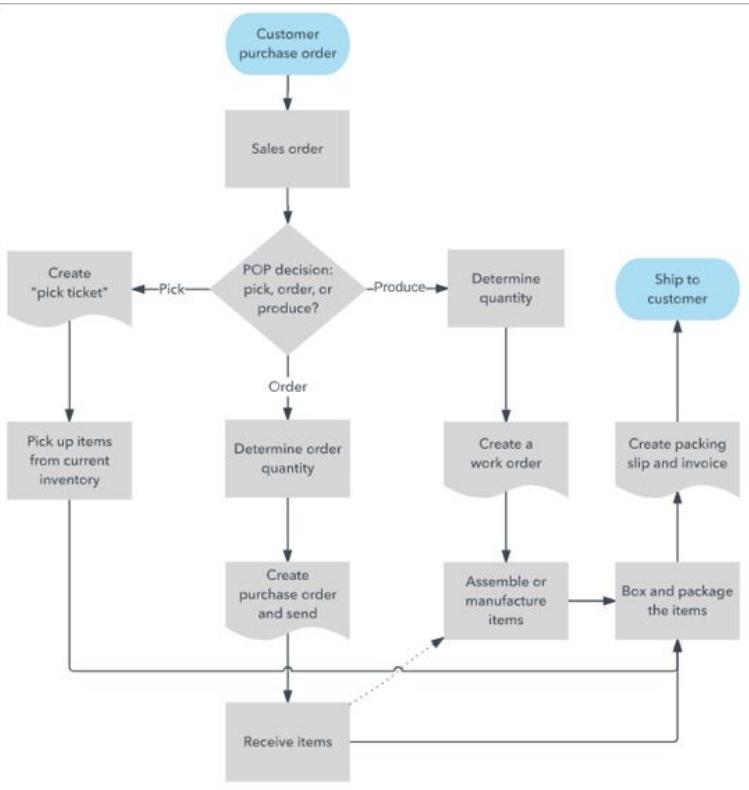
Fim-Enquanto

Imprimir melhor individuo

Fim Algoritmo Evolutivo

```
1: Procedimento Busca Local
2:   melhoria = 1
3:   Enquanto melhoria==1 faça
4:     Para i=1,...,p faça
5:       Para k=1,...,nclusters faça
6:         calcule coefik
7:         atualizar Max(coefik)
8:       Fim-Para
9:     Fim-Para
10:    re-associar as partes de acordo com Max(coefik),  $\forall i=1,\dots,p$ 
11:    Se solução não melhorou então
12:      associar partes à clusters originais
13:      melhoria = 0
14:    Fim-Se
15:    Se melhoria==1 então
16:      Para i=1,...,m faça
17:        Para k=1,...,nclusters faça
18:          calcule coefik
19:          atualizar Max(coefik)
20:        Fim-Para
21:      Fim-Para
22:      reassociar as máquinas de acordo com Max(coefik),  $\forall i=1,\dots,m$ 
23:      Se solução não melhorou então
24:        associar máquinas à clusters originais
25:        melhoria = 0
26:      Fim-Se
27:    Fim-Se
28:  Fim-Enquanto
29: Fim Procedimento
```

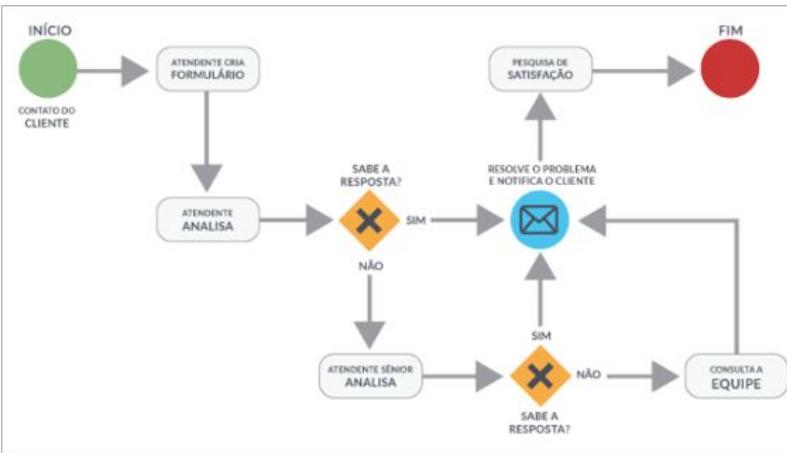
Exemplo de Fluxogramas



Existem inúmeros símbolos que podem ser utilizados para representar ações e decisões a serem tomadas durante seu processo. Separei alguns deles para que você possa visualizar de forma clara o que são esses símbolos.



Esses símbolos fazem parte de um padrão, como foi dito acima, permitindo o fácil entendimento do processo por parte daqueles que irão efetuar possíveis mudanças e melhorias no processo.



Exemplos de Linguagens de Programação



As linguagens de programação podem se dividir em dois tipos:

- Baixo nível
- Alto nível

Acesse [S.O.S Tecnologia e Educação](#) para entender mais.

Exemplos de Linguagens de Programação

```
1 import 'package:flutter/material.dart';
2 import 'model/bd_suporte.dart';
3 import 'model/pessoa.dart';
4
5 void main() {
6     //Criando e inicializando um WidgetsFlutterBinding.
7     //Se um foi inicializado anteriormente, pelo menos implementará WidgetsBinding
8     WidgetsFlutterBinding.ensureInitialized();
9     //Rodando o App
10    runApp(AppBD());
11 }
12
13 class AppBD extends StatefulWidget {
14     @override
15     _AppBDState createState() => _AppBDState();
16 }
17
18 class _AppBDState extends State<AppBD> {
19     //Variável para controle de indentificação
20     int? idSelecionado;
21     //Variável controladora de entrada de dados
22     final entrada = TextEditingController();
23
24     @override
25     Widget build(BuildContext context) {
26         return MaterialApp(
27             home: Scaffold(
28                 appBar: AppBar(
29                     title: TextField(
30                         //Entrada do Usuário
31                         controller: entrada,
32                     ),
33                 ),
34                 body: Center(
35                     //Criar uma lista de Pessoas
36                     child: FutureBuilder<List<Pessoa>>(
37                         future: SuporteDb.instancia.pegarPessoas(),
38                         builder:
```

O exemplo ao lado apresenta uma codificação para dispositivos móveis utilizando a linguagem de programação Dart em conjunto com o framework Flutter



O que é o Python?

Python é uma linguagem de programação de alto nível, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem, como um todo, não é formalmente especificada. O padrão na prática é a implementação CPython.

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens. Devido às suas características, ela é utilizada, principalmente, para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web. Foi considerada pelo público a 3^a linguagem "mais amada", de acordo com uma pesquisa conduzida pelo site Stack Overflow em 2018 e está entre as 5 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk.

O nome Python teve a sua origem no grupo humorístico britânico Monty Python, criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome (em português, píton ou pitão).



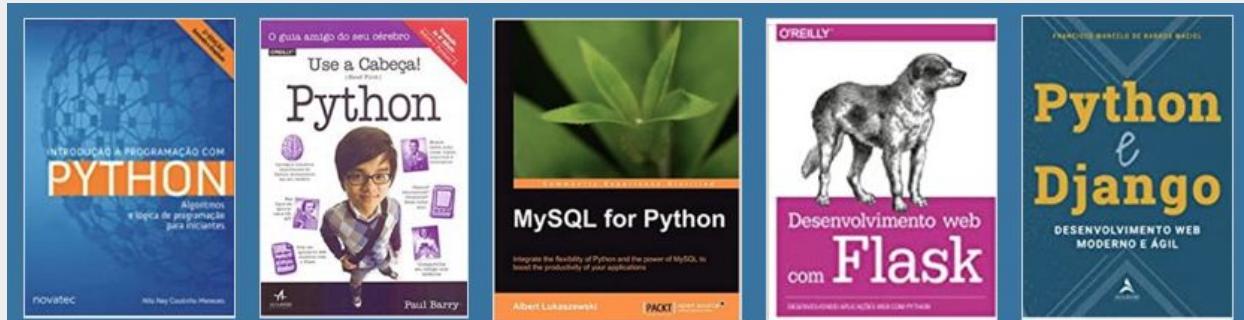
Importante

Python Brasil:

<https://python.org.br/>

Documentação

<https://docs.python.org/3/>



A linguagem

Banco de
Dados

Framework

Instalação do Python: [Download Python | Python.org](#)



python™

Donate



Search

GO

Socialize

About

Downloads

Documentation

Community

Success Stories

News

Events

Download the latest version for Windows

[Download Python 3.11.0](#)

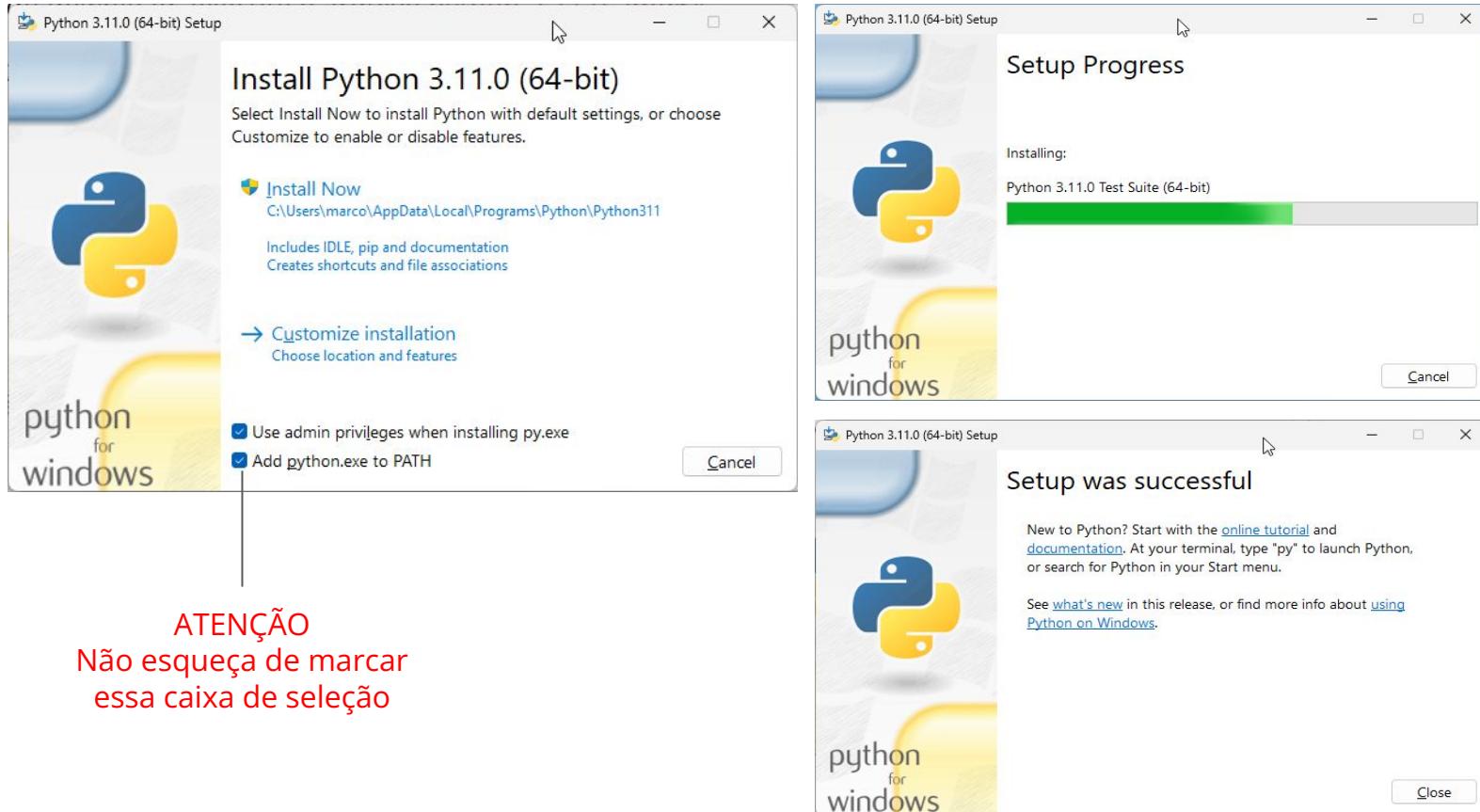
Looking for Python with a different OS? Python for [Windows](#),
[Linux/UNIX](#), [macOS](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#),
[Docker images](#)



Baixe a última versão do Python do site python.org. Siga as telas e quando aparecer uma mensagem para acrescentar o programa ao **path**, marque a opção.

Depois do download, execute o arquivo **python-3.11.0-amd64**



Instalação do Python: [Download Visual Studio Code - Mac, Linux, Windows](#)



Docs Updates Blog API Extensions FAQ Learn

Search Docs

Download

Download Visual Studio Code

Free and built on open source. Integrated Git, debugging and extensions.



↓ Windows

Windows 8, 10, 11



↓ .deb

Debian, Ubuntu

↓ .rpm

Red Hat, Fedora, SUSE



↓ Mac

macOS 10.11+

User Installer
System Installer
.zip

64 bit 32 bit ARM
64 bit 32 bit ARM
64 bit 32 bit ARM

.deb
.rpm
.tar.gz

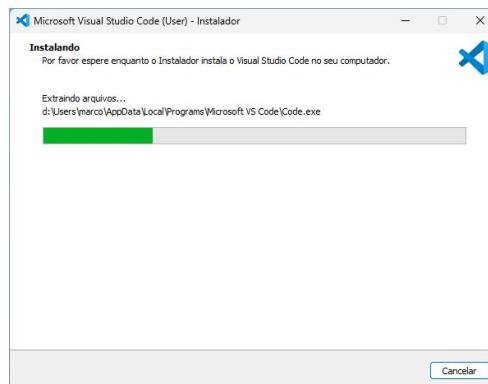
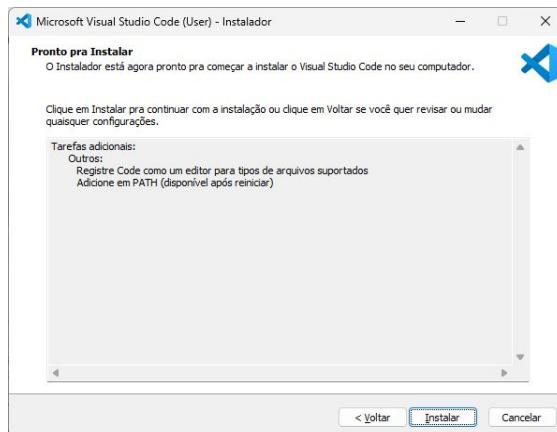
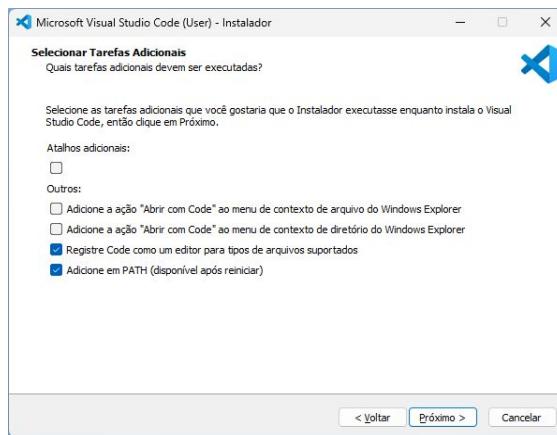
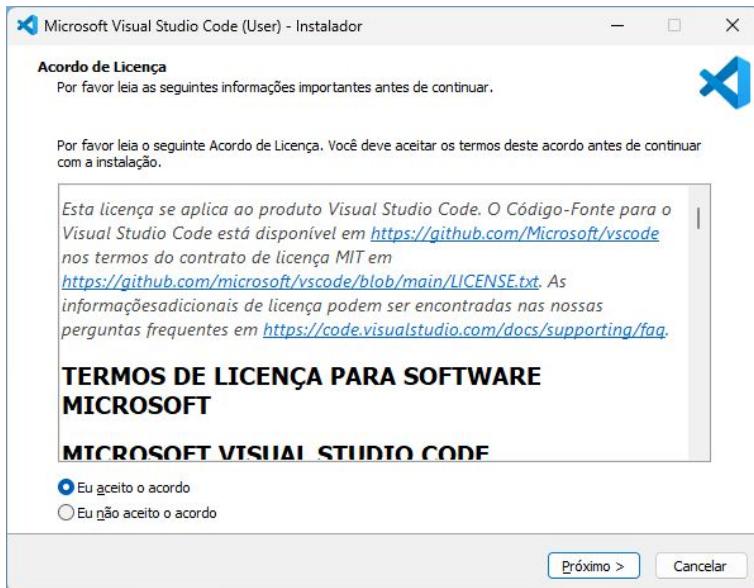
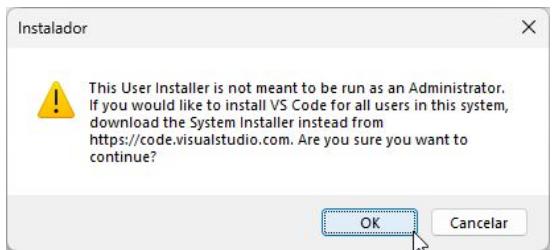
64 bit ARM ARM 64
64 bit ARM ARM 64
64 bit ARM ARM 64

.zip Universal Intel Chip Apple Silicon

Snap Store

Baixe a última versão do Vs Code do site code.visualstudio.com.
Siga as telas e até o término da instalação.
Depois de instalado, **acrescente a extensão do Python**

Depois do download, execute o arquivo **VSCodeUserSetup-x64-1.73.0**



O que é editor de código-fonte?

O editor de código-fonte é um software feito para uso de programadores para escreverem suas linhas de código.

Alguns deles são específicos para uma linguagem de programação, enquanto outros servem a várias tecnologias diferentes. Você pode baixar na sua máquina, mas também há os editores on-line ou em nuvem. [Exemplo: Atom, Sublime, Wordpad++ etc.](#)

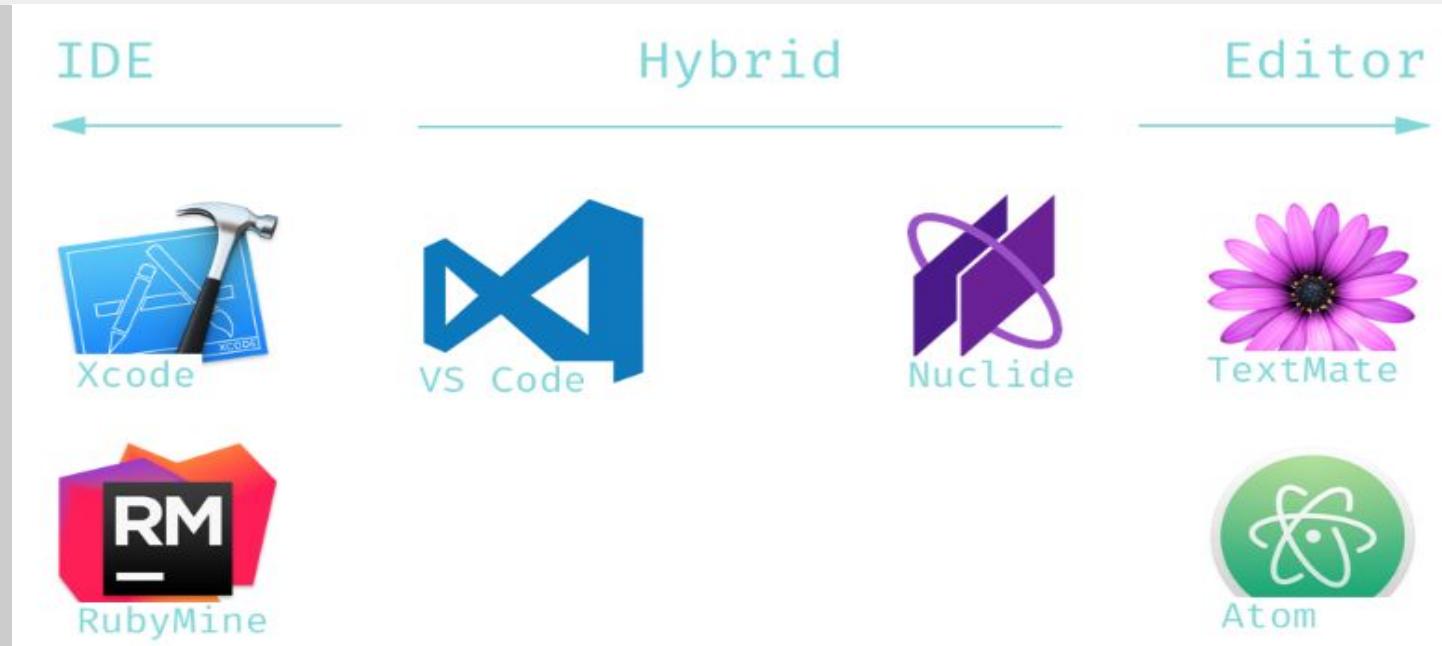
O que é um IDE?

É uma solução mais completa que a oferecida pelo editor de código-fonte. Um IDE é um Integrated Development Environment ou, simplesmente, Ambiente de Desenvolvimento Integrado.

Na prática, um IDE é um software que combina várias ferramentas de desenvolvimento em uma única interface gráfica do usuário. Em resumo, um IDE pode reunir um editor de código-fonte, a automação da compilação local e um debugger. Saiba em detalhes o que é IDE no nosso blog.

[Exemplo: Visual Studio Code, PyCharm, Eclipse etc.](#)

IDEs e Editores de Código: vantagens e desvantagens



The screenshot shows the Visual Studio Code interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates the file is 'primeiro_progama.py' in Python mode, running as an Administrator. The left sidebar contains the Explorer, Search, Source Control, Run and Debug, Extensions, Testing, Run Code, and Manager sections, each with its own icon. The main area displays the Python code for 'primeiro_progama.py'. The bottom status bar shows the file path, line and column numbers (Ln 20, Col 1), spaces used, encoding (UTF-8 CRLF), language (Python), and version (3.11.0 64-bit). A bottom navigation bar includes icons for back, forward, and search.

```
# Curso Técnico de Informática
# Autor: Sebastião Marcos
# Data: 03/11/2022
# Primeiro Programa em Python

# Importando as bibliotecas
import os

# Limpa o terminal
os.system('cls')

# O comando print() executa uma saída via terminal
print('*'*70)
print('Hello World!!!!')
print('*'*70)
print()
print()
```

Organização das Pastas (minha forma é opcional)

- Crie uma pasta Chamada '['projetos_python'](#)' ou só '['python'](#)'
- Dentro desta pasta crie uma subpasta para o projeto do dia. Ex.: [aula001_03_10_2022](#)
- Dentro da subpasta criar os arquivos que são pertinentes à aula

The screenshot shows the VS Code interface with the following details:

- EXPLORER View:** Shows a file tree structure:
 - Python folder (`.vscode`)
 - aula001_22-10-21_primeiroprograma** folder
 - `primeiro_progama.py`
 - aula002_25-10-21_variaveis** folder
 - `variaveis.py`
 - `aula003_26-10-21_entrada_de_dados`
 - `entrada.py`
 - `operadores_aritmeticos.py`
 - `precedencia.py`
 - `saida_formatada.py`
 - `teste.py`
 - `aula004_27-10-21_operadores_aritmeticos`
 - `operadores_aritmeticos.py`
 - Editor View:** Displays the content of `primeiro_progama.py`:

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 03/11/2022
4 # Primeiro Programa em Python
5
6 # Importando as bibliotecas
7 import os
8
9
10 # Limpando o terminal
11 os.system('cls')
12
13 # O comando print() executa uma saída via terminal
14
15 print('*'*70)
```

Considerações PEP8

O PEP - 8 é um manual de boas práticas de escrita de código Python que visa padronizar uma escrita de código / comentários, dessa forma "Embelezando" o código e tornando-o mais legível!! Várias empresas adotam o PEP - 8 em suas equipes para facilitar o trabalho em grupo.

Consideração 1: Indentar o código

```
1 # Aligned with opening delimiter.
2 foo = long_function_name(var_one, var_two,
3 | | | | | var_three, var_four)
4
5 # Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
6 def long_function_name(
7     var_one, var_two, var_three,
8     var_four):
9     print(var_one)
10
11 # Hanging indents should add a level.
12 foo = long_function_name[
13     var_one, var_two,
14     var_three, var_four]
```

Consideração 2: tamanho máximo para a coluna de caracteres menor ou igual a 80

```
1 # Aligned with opening delimiter.
2 foo = long_function_name(var_one, var_two,
3 | | | | | var_three, var_four)
4 |
5 # Add 4 spaces (an extra level of indentation) to distinguish arguments from the rest.
6 def long_function_name(
7     var_one, var_two, var_three,
8     var_four):
9     print(var_one)
10
11 # Hanging indents should add a level.
12 foo = long_function_name(
13     var_one, var_two,
14     var_three, var_four)
```

Consideração 3:

- Linhas em Branco:

1 linha depois de classes

2 linhas depois de métodos/funções importantes

1 linha para separar lógicas de código.

O import da biblioteca principal em primeiro lugar + espaço + as bibliotecas de terceiros

```
1  from time import sleep
2  import random
3
4  import pytz
5
6  def contador (i, f, p):
7      # Se o passo for negativo ou igual a zero é preciso tratá-lo
8      if p < 0:
9          p *= -1
10     if p == 0:
11         p = 1
12
13
14     print(f'Contagem de {i} até {f} de {p} em {p}')
15     sleep(1)
16
```

Consideração 4 e 5:

- cada import deve ser feito em uma linha
- espaços dentro do código

```
1 from time import sleep
2 import random
3
4 import pytz
5
6 def contador (i, f, p):
7     # Se o passo for negativo ou igual a zero é preciso tratá-lo
8     if p == 0:
9         raise ValueError('Passo deve ser maior que zero')
10    if f < i:
11        raise ValueError('O final deve ser maior que o inicio')
12    else:
13        while i <= f:
14            sleep(0.1)
15            print(i)
16            i += p
```

```
1 # CERTO:
2 spam(ham[1], {eggs: 2})
3 # ERRADO:
4 spam( ham[ 1 ], { eggs: 2 } )
```

Consideração 6:

- nome de classes, funções, variáveis e CONSTANTES:

```
6  MINHA_CONSTANTE = 0
7  minha_variavel = 0
8
9  def contador (i, f, p):
10     # Se o passo for negativo ou igual a zero é preciso tratá-lo
11     if p < 0:
12         p *= -1
13     if p == 0:
14         p = 1
15
```

- **Observação:** não utilizamos underline para declarar nomes compostos para as classes

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 03/11/2022
4 # Primeiro Programa em Python
5
6 # Importando as bibliotecas
7 import os
8
9 # Limpando o terminal
10 os.system('cls')
11
12 # O comando print() executa uma saída via terminal
13
14 print('*'*70)
15 print('Hello World!!!')
16 print('*'*70)
17 print()
18 print()

```

Saída com Code Runner

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

[Running] python -u "d:\Trabalho\Senac\Turma 0277\UC10\python\aula001_22-
FF
Hello World!!!

[Done] exited with code=0 in 0.042 seconds

Saída com comando python em terminal

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Hello World!!!

PS D:\python\aula_001_primeiro_programa> []

Comando/Função para imprimir
alguma coisa na tela

Exemplos do comando print

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpa o terminal
9 os.system('cls')
10
11 print('Olá mundo')
12 print("Olá mundo!")
13
14 # imprimindo a string com as simples e duplas
15 print('Olá mundo')
16 print("Olá mundo!")
```

```
2
3 print("Olá Mundo!")
4
```

No Python todos os comandos são
funções e essas funções possuem o
parênteses como delimitador.

Se os dados tiverem mensagens
é preciso utilizar delimitadores,
nesse caso aspas simples ou
duplas

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
PS C:\Users\marco\OneDrive\Área de Trabalho\Python> & do.py"
```

```
Olá Mundo!
```

```
Estamos estudando a liguagem:  
PYTHON!
```

```
PS C:\Users\marco\OneDrive\Área de Trabalho\Python>
```

Print para Imprimir uma mensagem na tela

```
1 print("Olá Mundo!")  
2 print('Estamos estudando a liguagem:')  
3 print('PYTHON!')  
4
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
PS C:\Users\marco\OneDrive\Área de Trabalho\Python> & do.py"
```

```
2
```

```
10
```

```
10000
```

```
PS C:\Users\marco\OneDrive\Área de Trabalho\Python>
```

Print para imprimir um cálculo na tela

```
1 print(1 + 1)  
2 print(40 - 30)  
3 print([200 * 50])
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

Variáveis são um dos recursos mais básicos das linguagens de programação. Utilizadas para armazenar valores em memória, elas nos permitem gravar e ler esses dados com facilidade a partir de um nome definido por nós. Neste documento aprenderemos a declarar e atribuir valores a variáveis em Python.

Exemplo: precisamos guardar o ano de nascimento de um usuário e o ano atual. Com esses valores podemos fazer uma operação de subtração para encontrar a idade deste mesmo usuário.

Observação: a medida que formos avançando veremos outros tipos importantes de variáveis

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # Declarações
12 nome = 'John Doe'
13 data_nascimento = 1970
14 altura = 1.77
15 peso = 72.5
16 doador = True
17 complexo = 3j
18 CONSTANTE = 10
19
20 # Saída
21 print('*'*70)
22 print('ESTUDO DE VARIÁVEIS: TIPOS')
23 print('*'*70)
24 print('A variável nome é do tipo ', type(nome))
25 print('A variável data_nascimento é do tipo ', type(data_nascimento))
26 print('A variável altura é do tipo ', type(altura))
27 print('A variável peso é do tipo ', type(peso))
28 print('A variável doador é do tipo ', type(doador))
29 print('A variável complexo é do tipo ', type(complexo))
30 print('A variável CONSTANTE é do tipo ', type(CONSTANTE));
31
32 print('*'*70)
33
34 print()
35 print('*'*70)
36 print('ESTUDO DE VARIÁVEIS: SAÍDA')
37 print('*'*70)
38 print('Seu nome é.....: ', nome)
39 print('Ano de nascimento.....: ', data_nascimento)
40 print('Você mede.....: ', altura, 'm')
41 print('Seu peso é.....: ', peso, 'Kg')
42 print('Impressão de número complexo..: ', complexo)
43 print('Impressão de uma constante....: ', CONSTANTE)
44
45 print('*'*70)
46 print('Fim do Programa!')
47 print('*'*70)
```

ESTUDO DE VARIÁVEIS: TIPOS
=====

```
A variável nome é do tipo <class 'str'>
A variável data_nascimento é do tipo <class 'int'>
A variável altura é do tipo <class 'float'>
A variável peso é do tipo <class 'float'>
A variável doador é do tipo <class 'bool'>
A variável complexo é do tipo <class 'complex'>
A variável CONSTANTE é do tipo <class 'int'>
```

ESTUDO DE VARIÁVEIS: SAÍDA
=====

```
Seu nome é.....: John Doe
Ano de nascimento.....: 1970
Você mede.....: 1.77 m
Seu peso é.....: 72.5 Kg
Impressão de número complexo...: 3j
Impressão de uma constante....: 10
```

=====

Fim do Programa!

String (str): Na programação String representa um conjunto de caracteres disposto numa determinada ordem. A partir de agora, todas as vezes em que falarmos o termo String, estaremos nos referindo a um conjunto de caracteres.

Inteiro (int): Números inteiros tem uma representação fixa e não possuem casas decimais. São utilizados para representar valores, como por exemplo, ano de nascimento, dia da semana, idade... Essa notação pode não ser a mais utilizadas em livros ou apostilas, mas vai te dar uma noção mais clara da funcionalidade desse tipo.

Float: Números float, também conhecidos como ponto flutuante. No Python é utilizado para representar números com casas decimais. Utilizamos esse tipo em: valor salarial, altura, peso, cálculo de área etc. Essa notação pode não ser a mais utilizadas em livros ou apostilas, mas vai te dar uma noção mais clara da funcionalidade desse tipo.

Boolean (bool): Valores geralmente utilizados para a implementação lógica dentro de condicionais ou no uso de flags para segurar a execução de parte do código. Esses valores são verdadeiros ou falsos.

complex: Tipo de dado usado para representar números complexos. Esse tipo normalmente é usado em cálculos geométricos e científicos. Um tipo complexo contém duas partes: a parte real e a parte imaginária, sendo que a parte imaginária contém um j.

- Uma variável pode ter um nome curto (como id) ou um nome mais descritivo (idade, nomealuno, valor_total)
- Um nome de variável deve começar com uma letra ou o caractere de sublinhado
- O nome de uma variável não pode começar com um número
- Um nome de variável pode conter apenas caracteres alfanuméricos e sublinhados (Az, 0-9 e _)
- Os nomes das variáveis diferenciam maiúsculas de minúsculas (idade, Idade e IDADE são três variáveis diferentes)

Algumas técnicas para criação de variáveis

Camel case: minhaVariavel, nomeAluno

Pascal case: MinhaVariavel, NomeAluno

Snake case: minha_variavel, nome_aluno (**PYTHON**)

A função `input()` recebe como parâmetro uma string que será mostrada como auxílio ao usuário, geralmente informando que tipo de dado o programa está aguardando receber. Após a entrada de dados pelo usuário, o programa irá continuar sua execução, passando para as instruções seguintes.

Variável que irá receber a
entrada do usuário

Operador de atribuição

```
1 nome = str(input('Qual o seu nome:'))
```

`str` é o tipo de dado que a variável vai
receber, nesse caso uma `String` (Cadeia
de texto)



Aqui acontece um casting

Texto informativo
para o usuário

! Na programação Python,
utilizamos o comando
`INPUT()` para receber
dados do usuário

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

PS C:\Users\marco\OneDrive\Área de Trabalho\Python>
Qual o seu nome: █

Variável que irá receber a
entrada do usuário

```
1  
2  idade = int(input('Qual a sua idade:'))  
3
```

int é o tipo de dado que a variável vai receber,
nesse caso um **Inteiro** [...] -2, -1, 0, 1, 2...[

Texto informativo para o
usuário

Na programação Python,
utilizamos o comando
INPUT() para receber
dados do usuário



OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

PS C:\Users\marco\OneDrive\Área de Trabalho\Python>
Qual a sua idade:■

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Uso do comando input
6
7 # Entrada _____
8 nome = str(input('Qual o seu nome: '))
9 dataNascimento = int(input('Em que ano você nasceu: '))
10 altura = float(input('Qual a sua altura: '))
11 peso = float(input('Quanto você pesa: '))
12
13
14 #Saída _____
15 print('*'*70)
16 print('ESTUDO DE ENTRADA DE DADOS COM INPUT')
17 print('*'*70)
18 print('Seu nome é.....: ', nome)
19 print('Ano de nascimento.....: ', dataNascimento)
20 print('Você mede.....: ', altura, 'm')
21 print('Seu peso é.....: ', peso, 'Kg')
22 print('*'*70)
23 print('Fim do Programa!')
24 print('*'*70)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Qual o seu nome: John Doe
Em que ano você nasceu: 1970
Qual a sua altura: 1.77
Quanto você pesa: 80

ESTUDO DE ENTRADA DE DADOS COM INPUT

=====

Seu nome é.....: John Doe
Ano de nascimento.....: 1970
Você mede.....: 1.77 m
Seu peso é.....: 80.0 Kg

=====

Fim do Programa!

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # Entrada _____
12 nome = str(input('Nome completo: '))
13 endereço = str(input('Digite o endereço: '))
14 cep = int(input('Entre com o cep: '))
15 cidade = str(input('Digite o nome da cidade: '))
16 estado = str(input('Entre com o estado: '))
17 nota_enem = float(input('Digite sua nota no Enem: '))
18
19
20 # Saída Formatada _____
21 print('*'*70)
22 print('ESTUDO DE SAÍDA FORMATADA')
23 print('*'*70)
24 print(f'Nome completo: {nome}')
25 print(f'Endereço: {endereço}, ')
26 print(f'CEP: {cep}')
27 print(f'Cidade: {cidade} ')
28 print(f'Estado: {estado} ')
29 print(f'Nota no Enem: {nota_enem:.2f} ')
30 print('*'*70)
31 print('Fim do Programa!')
32 print('*'*70)
```

Nome completo: John Doe
Digite o endereço: Av Rio Branco, 1300
Entre com o cep: 36036630
Digite o nome da cidade: Juiz de Fora
Entre com o estado: MG
Digite sua nota no Enem: 800.5

ESTUDO DE SAÍDA FORMATADA

Nome completo: John Doe
Endereço: Av Rio Branco, 1300
CEP: 36036630
Cidade: Juiz de Fora
Estado: MG
Nota no Enem: 800.50

Fim do Programa!

Utilizando o que foi aprendido até o momento, faça um programa que receba e imprima os dados de 2 usuários seguindo a listagem abaixo. Para Entrada utilizar casting e a saída deverá ser formatada.

Dados Pessoais

Nome

Data de nascimento

RG

CPF

Endereço

Rua

Número

Complemento

CEP

Cidade

Estado

País

Forma de envio da atividade:

- envie o arquivo python (.py)
- e-mail: profsebastiaomarcos@gmail.com
- assunto: Python – Atividade 001 – Seu nome
- Não esqueça de anexar o arquivo
- Para mais de 1 arquivo, compactar e utilizar o mesmo procedimento.

Os operadores aritméticos são usados quando precisamos realizar alguma operação matemática em nosso código. Essas operações, na maioria dos casos, são atribuídas a uma variável que receberá o valor da operação.

+	Adição	10	+	2	12
-	Subtração	10	-	2	8
*	Produto	10	*	2	20
/	Divisão	10	/	2	5
**	Potência	5	**	2	25
//	Divisão Inteira	9	//	4	2
%	Resto da Divisão ou Módulo	9	%	4	1

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # declaração
12 a = 10
13 b = 2
14
15 # Cálculos
16 # soma
17 soma = a + b
18 # Subtração
19 subtracao = a - b
20 # produto
21 produto = a * b
22 # divisao
23 divisao = a / b
24 # Potência
25 potencia = a ** b
26 # Divisão Inteira
27 divisaoInteira = a // b
28 # Resto da divisão
29 restoDivisao = a % b
30

31 # Saída
32 print('-'*70)
33 print('ESTUDO DE OPERADORES ARITMÉTICOS')
34 print('='*70)
35 print(f'A soma de {a} + {b} = {soma}')
36 print(f'A subtração de {a} - {b} = {subtracao}')
37 print(f'A multiplicação de {a} x {b} = {produto}')
38 print(f'O divisão de {a} / {b} = {divisao:.4f}')
39 print(f'O número {a} elevado a {b} = {potencia}')
40 print(f'A divisão inteira de {a} por {b} = {divisaoInteira}')
41 print(f'O resto da divisão de {a} por {b} = {restoDivisao}')
42 print('raiz quadrada: (TENTE RESOLVER)')
43 print('raiz cúbica: (TENTE RESOLVER)')
44 print(f'-'*70)

```

ESTUDO DE OPERADORES ARITMÉTICOS

```

A soma de 10 + 2 = 12
A subtração de 10 - 2 = 8
A multiplicação de 10 x 2 = 20
O divisão de 10 / 2 = 5.0000
O número 10 elevado a 2 = 100
A divisão inteira de 10 por 2 = 5
O resto da divisão de 10 por 2 = 0
raiz quadrada: (TENTE RESOLVER)
raiz cúbica: (TENTE RESOLVER)

```

Utilizando os conhecimentos dos slides anteriores, monte um estudo codificado sobre operadores aritméticos utilizando entradas de dados com saídas formatadas. Além das operações realizadas, acrescente raiz quadrada e raiz cúbica.

Forma de envio da atividade:

- envie o arquivo python (.py)
- e-mail: profsebastiaomarcos@gmail.com
- assunto: Python – Atividade 002 – Seu nome
- Não esqueça de anexar o arquivo
- Para mais de 1 arquivo, compactar e utilizar o mesmo procedimento.

Exemplo da saída:

ESTUDO DE OPERADORES ARITMÉTICOS

Informe o 1º valor: 4
Informe o 2º valor: 8

A soma de 4.0 + 8.0 = 12.0
A subtração de 4.0 - 8.0 = -4.0
A multiplicação de 4.0 x 8.0 = 32.0
O divisão de 4.0 / 8.0 = 0.5000
O número 4.0 elevado a 8.0 = 65536.0
A divisão inteira de 4.0 por 8.0 = 0.0
O resto da divisão de 4.0 por 8.0 = 4.0
raiz quadrada de 4.0: 2.0
raiz cubica de 8.0: 2.0

Fim do Programa!

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # Entrada
12 nota1 = float(input('1ª nota: '))
13 nota2 = float(input('2ª nota: '))
14 nota3 = float(input('3ª nota: '))
15 nota4 = float(input('4ª nota: '))
16
17 # operações
18 soma = nota1 + nota2 + nota3 + nota4
19 media = nota1 + nota2 + nota3 + nota4 / 4
20 mediaCorreta = (nota1 + nota2 + nota3 + nota4) / 4
21
22 # Saída
23 print(f'1ª nota do aluno: {nota1}')
24 print(f'2ª nota do aluno: {nota2}')
25 print(f'3ª nota do aluno: {nota3}')
26 print(f'4ª nota do aluno: {nota4}')
27 print(f'A soma das notas é: {soma}')
28 print(f'A média: {media} ERRADO!')
29 print(f'A média: {mediaCorreta} CORRETO')

```

Os operadores aritméticos são usados quando precisamos realizar alguma operação matemática em nosso código. Essas operações, na maioria dos casos, são atribuídas a uma variável que receberá o valor da operação.

1º	()	Parênteses
2º	**	Potência
3º	* / // %	Produto – Divisão – Divisão inteira – Módulo
4º	+	Adição – Subtração

Importante!

- 1º caso, não tem [] ou { }, eles exercem outras funções no python
- 3º caso, quem vier primeiro
- 4º caso, quem vier primeiro

Resto divisão

- a. Faça um programa que peça 3 valores , depois calcule e imprima a soma e a multiplicação desses valores.
- b. Faça um programa que peça o ano do seu nascimento e calcule a sua idade.
- c. Faça um programa que peça 4 notas, após a entrada de dados calcular a média das notas digitadas.
- d. Faça um programa que receba e divida 2 números. A saída da divisão precisará ser formatada com 4 casas decimais.
- e. Faça um programa que recebe um número inteiro e mostre o sucessor e antecessor.
- f. Faça um programa que receba um número qualquer e calcule o dobro e o triplo desse número.
- g. Faça um programa que converta metros em centímetros e milímetros.
- h. Faça um programa que receba um número inteiro, depois imprima sua tabuada de multiplicação.
- i. Faça um programa que receba um valor em reais, depois calcule quantos dólares daria para comprar com esse valor.
- j. Faça um programa com entrada de dados para calcular o perímetro de um retângulo.

Observação

É obrigatório a saída formatada!

O código precisa ter o cabeçalho padrão
o código precisa ser comentado

Enviar somente os exercícios indicados com a cor laranja.

No Python utilizamos o comando **IF** para atender os desvios condicionais em nosso programa. Quando programamos, muitas vezes precisamos que determinado bloco de código seja executado apenas se uma determinada condição for verdadeira. Em casos assim, devemos fazer uso de uma estrutura condicional.

O comando **IF** é uma estrutura condicional que permite avaliar uma expressão e, de acordo com seu resultado, executar uma determinada ação.

Mas nossa condicional pode também cair em um resultado falso, neste caso, utilizamos o comando **ELSE** para exibir o resultado da condição.

A seguir demonstraremos essas variações com a utilização dos comandos **IF** e **ELSE** dentro do código Python. Também abordaremos algumas particularidades dos comandos **IF ELSE** em mais de um teste, ou seja, a utilização de um comando **ELIF**. Nesse caso, testamos mais de uma vez a condição, caso haja mais operações para serem analisadas.

Na tabela ao lado, temos os **operadores relacionais** disponibilizados pelo Python para codificação em conjunto com as condicionais. O Python utiliza o recuo (espaço em branco no início de uma linha) para definir o escopo do código, ou seja, o bloco de código que será executado com aquela instrução. Outras linguagens de programação geralmente usam as chaves para definir o bloco de código. Chamamos esse recurso de indentação.

```

1 # Declaração
2 condicao = True
3
4 # teste condicional
5 if (condicao):
6     # verdadeiro
7     pass
8 # testa condicional
9 elif condicao:
10    # Verdadeiro
11    pass
12 # Caso contrário
13 else:
14    pass

```

<code>==</code>	Igual a
<code>!=</code>	Diferente
<code>>=</code>	Maior ou igual
<code>></code>	Maior que
<code><</code>	Menor que
<code><=</code>	Menor ou igual

Além dos operadores relacionais, existem os chamados operadores lógicos ou "conectivos lógicos". Estes, servem para conectar/combinar duas expressões relacionais. Nessa parte seria muito interessante um estudo de lógica matemática para agregar complexidade às relações condicionais do seu código.

or	OU lógico
and	E lógico
not	Negação

Tabela verdade para os operadores relacionais and, or e not

A	B	and	or
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

A	not
True	False
False	True

```

1 condicao1 = True
2 condicao2 = False
3
4 if ((condicao1 == True and condicao2 == False) or
5     (condicao1 != False and condicao2 != True)):
6     print('Verdadeiro: Imprima esse bloco!')
7 else:
8     print('Falso: Imprima esse bloco!')

```



Praticando...

Faça um programa que peça um valor decimal ao usuário. Em seguida, se o usuário digitado for um valor inteiro, o programa deverá mostrar uma mensagem de "valor inválido!".

Depois do código executado e testado, o que aconteceu?

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 print('*'*50)
12 print('Prática: verificando valor quebrado')
13 print('*'*50)
14
15 # Entrada de dados
16 valor = float(input('Digite um valor com casas decimais: '))
17
18 # Condicional
19 if (valor % 2 == 0):
20     print(f'Valor {valor} inválido! O número digitado é inteiro')
21 else:
22     print()
23     print(f'O valor digitado foi {valor}, entrada correta!')
24
25 print('*'*50)
26 print('Fim do programa!')
27 print('*'*50)
```

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # Declaração
12 a = 10
13 b = 5
14 c = 'John'
15
16 # Condicional Simples
17 print('*'*40)
18 print('CONDICIONAL SIMPLES')
19 print('*'*40)
20 if (a > b):
21     print('"a" é maior que "b"!')
22 else:
23     print('"a" não é maior que "b"')
24 print()
25
26 # Condicional Aninhada
27 print('*'*40)
28 print('CONDICIONAL ANINHADA')
29 print('*'*40)
30 if (a < b):
31     print('"a" é maior que "b"!')
32 elif (b != 5):
33     print('"b" é diferente de "c"')
34 elif (c == 'John'):
35     print('"c" é igual a "John"')
36 else:
37     print('Se nada deu certo!')
38 print()
```

CONDICIONAL SIMPLES

"a" é maior que "b"!

CONDICIONAL ANINHADA

"c" é igual a "John"

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 a = 10
12 b = 5
13 c = 'José'
14 d = 'José'
15
16 print('*'*40)
17 print('ESTUDO DE CONDICIONAIS: OPERADORES RELACIONAIS ')
18 print()
19 # Operador == (igualdade)
20 if (c == d):
21     print('*'*40)
22     print(f'{c} é igual a {d}')
23     print('*'*40)
24 else:
25     print(f'{c} não é igual a {d}')
26
27 # Operador != (diferença)
28 if (a != c):
29     print('*'*40)
30     print(f'{a} é diferente a {c}')
31     print('*'*40)
32 else:
33     print(f'{a} não é diferente a {c}')
34
35 # Operador > (maior que)
36 if (a > b):
37     print('*'*40)
38     print(f'{a} é maior que {b}')
39     print('*'*40)
40 else:
41     print(f'{a} não é maior que {b}')
42
43 # Operador < (menor que)
44 if (b < a):
45     print('*'*40)
46     print(f'{b} é menor que {a}')
47     print('*'*40)
48 else:
49     print(f'{b} não é menor que {a}')
50
51 #Operador >= (maior ou igual)
52 if (a >= b):
53     print('*'*40)
54     print(f'{a} é maior ou igual que {b}')
55     print('*'*40)
56 else:
57     print(f'{a} não é maior ou igual a {b}')
58
59 #Operador <= (menor ou igual)
60 if (b <= a):
61     print('*'*40)
62     print(f'{b} é menor ou igual que {a}')
63     print('*'*40)
64 else:
65     print(f'{b} não é menor ou igual a {a}')

```

ESTUDO DE CONDICIONAIS: OPERADORES RELACIONAIS

José é igual a José

10 é diferente a José

10 é maior que 5

5 é menor que 10

10 é maior ou igual que 5

5 é menor ou igual que 10

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # Declaração
12 a = 10
13 b = 5
14 c = 'John'
15
16 print('*'*40)
17 print('ESTUDO DE CONDICIONAIS: OPERADORES LÓGICOS')
18 print('*'*40)
19 print()
20
21 # E (and) VERDADEIRO
22 print('E (and) VERDADEIRO')
23 if (a > 5 and b != c):
24     print('Verdadeiro: Bloco executado')
25 else:
26     print('Falso: Bloco executado')
27
28 print()
29
30 # E (and) FALSO
31 print('E (and) FALSO')
32 if (a > 5 and b == c):
33     print('Verdadeiro: Bloco executado')
34 else:
35     print('Falso: Bloco executado')
36
37 print()
38
39 # OU (or) VERDADEIRO
40 print('OU (or) VERDADEIRO')
41 if (a > 5 or b == c):
42     print('Verdadeiro: Bloco executado')
43 else:
44     print('Falso: Bloco executado')
45
46 print()
47
48 # OU (or) FALSO
49 print('OU (or) FALSO')
50 if (a < 5 or c == 'Jane'):
51     print('Verdadeiro: Bloco executado')
52 else:
53     print('Falso: Bloco executado')

```

ESTUDO DE CONDICIONAIS: OPERADORES LÓGICOS

E (and) VERDADEIRO
Verdadeiro: Bloco executado

E (and) FALSO
Falso: Bloco executado

OU (or) VERDADEIRO
Verdadeiro: Bloco executado

OU (or) FALSO
Falso: Bloco executado

- a. Faça um programa que peça um número inteiro. Depois verifique se esse número é par ou se ele é ímpar.
- b. Faça um programa que peça ao usuário 3 números. Depois, mostre na tela qual é o maior, o menor número ou se os números são iguais.
- c. Faça um programa que peça a velocidade de um carro. Se o carro ultrapassar o limite de 60Km por hora, mostre na tela a mensagem: 'Limite de velocidade acima do permitido'. Se a velocidade estiver abaixo dos 60Km, mostrar a frase: 'Velocidade normal! A velocidade não pode ser negativa.'
- d. Faça um programa que calcule um aumento salarial de um funcionário. Se o salário for maior que R\$1500,00, efetuar um aumento de 5%; Se o salário for menor que R\$1000,00, efetuar um aumento de 10%. O salário não pode ser 0 ou negativo.
- e. Faça um programa que peça para o usuário a distância de uma viagem. Calcule o preço da passagem sabendo que a empresa de ônibus cobra R\$0,70 por km rodado, em viagens de até 200Km. Acima dessa distância as viagens passam a custar R\$0,40 por km rodado.
- f. Faça um programa que peça ao usuário um ano aleatório. Depois, mostre na tela se o ano é bissexto. Faça a validação de entrada de dados para valores menores ou iguais a 0.
- g. Faça um programa que calcule a formação de um triângulo. Para isso, o programa deverá receber a medida de 3 segmentos e, em seguida, compará-los. A resposta dessa comparação deverá ser: 'Forma um triângulo' e 'não forma um triângulo'. Os valores não podem ser 0 ou menor.
- h. Faça um programa que calcule a equação $x^2 - 6x + 5$. As raízes são {5, 1}. Esse cálculo deverá ser feito sem ajuda de funções ou métodos. Somente utilizando o que foi aprendido até o momento.

Observação

É obrigatório a saída formatada!

O código precisa ter o cabeçalho padrão

o código precisa ser comentado

Enviar somente os exercícios indicados com a cor laranja.

Enquanto a referência da Linguagem Python descreve a sintaxe e a semântica da programação, este manual de referência de bibliotecas descreve a biblioteca padrão que é distribuída com o Python. Ela também descreve alguns dos componentes opcionais que são comumente incluídos nas distribuições do Python.

A biblioteca padrão do Python é muito extensa, oferecendo uma ampla gama de recursos, conforme indicado pelo longo índice listado ao acessar o link. A biblioteca contém módulos internos (escritos em C) que fornecem acesso à funcionalidade do sistema, como E/S de arquivos que de outra forma seriam inacessíveis para programadores Python, bem como módulos escritos em Python que fornecem soluções padronizadas para muitos problemas que ocorrem em programação cotidiana. Alguns desses módulos são explicitamente projetados para incentivar e aprimorar a portabilidade de programas em Python, abstraindo os detalhes da plataforma em APIs neutras em plataforma.

Veja um pouco mais aqui:

[math — Função matemática — documentação Python](#)

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando toda a biblioteca da função
6 import math
7
8 # Especificando o import
9 from math import sqrt
10
11 import random
12 import os
13
14 # Limpando o terminal
15 os.system('cls')
16
17 # utilização
18
19 numero = 4
20
21 # para o import da biblioteca toda
22 raizQuadrada = math.sqrt(numero)
23
24 # para um import específico
25 raizQuadrada = sqrt(numero)
```

Esse módulo provê várias funções relacionadas à tempo. Para funcionalidades relacionadas veja também os módulos datetime e calendar

Apesar desse módulo sempre estar disponível, nem todas as suas funções estão disponíveis em todas as plataformas. A maioria das funções definidas nesse módulo chamam funções da biblioteca da plataforma de C com mesmo nome. Pode ser útil consultar a documentação da plataforma, pois a semântica dessas funções variam a depender da plataforma.

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 from datetime import datetime
12 from datetime import date
13
14 # Declarando uma variável data
15 data = datetime.now()
16
17 # Declarando uma variável data formatada
18 data_formatada = data.strftime('%d/%m/%Y')
19
20 # Declarando uma variável data e hora formatada
21 data_hora_formatado = data.strftime('%d/%m/%Y %H:%M')
22
23 print(f'Data formatada: {data_formatada}')
24 print(f'Data e hora formatadas: {data_hora_formatado}')
25
26 # recebendo o ano
27 data_atual = date.today()
28 nascimento = 1970
29
30 # Imprimindo só o ano
31 print(f'Ano atual: {data_atual.year}')
32
33 idade = data_atual.year - nascimento
34
35 # Imprimindo só o idade
36 print(f'Sua idade é: {idade} anos')
```

`math.ceil(x)`

Retorna o arredondamento para cima.

`math.floor(x)`

Retorna o arredondamento para baixo.

`math.pow(x, y)`

Retorna x elevado à potência y..

`math.sqrt(x)`

Retorna a raiz quadrada de x.

`math.cos(x)`

Retorna o cosseno de x radianos.

`math.sin(x)`

Retorna o seno de x radianos.

`math.tan(x)`

Retorna o tangente de x radianos.

`math.hypot(c_oposto, c_adjacente)`

Retorna o valor da hipotenusa

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7 import math
8
9 # Limpando o terminal
10 os.system('cls')
11
12 print('*'*50)
13 print('ESTUDO DA BIBLIOTECA MATEMÁTICA MATH')
14 print('*'*50)
15 print()
16 # Declarações
17 base = 2
18 expoente = 3
19 angulo = 30
20 radicando = 81
21 co = 5
22 ca = 10
23
24 # Cálculo
25 potencia = math.pow(base, expoente)
26 raizQuadrada = math.sqrt(radicando)
27 seno = math.sin(math.radians(angulo))
28 cosseno = math.cos(math.radians(angulo))
29 tangente = math.tan(math.radians(angulo))
30 hipotenusa = math.hypot(co, ca)
31
32 # Saída
33 print(f'{base} elevado a {expoente} é igual a: {potencia}')
34 print(f'A raiz quadrada de {radicando} é: {raizQuadrada}')
35 print(f'O seno de {angulo} é: {seno}')
36 print(f'O cosseno de {angulo} é: {cosseno}')
37 print(f'O seno de {angulo} é: {tangente}')
38 print(f'O valor da hipotenusa é: {hipotenusa}')
39 print('*'*50)

```

ESTUDO DA BIBLIOTECA MATEMÁTICA MATH

=====
2 elevado a 3 é igual a: 8.0
A raiz quadrada de 81 é: 9.0
O seno de 30 é: 0.4999999999999994
O cosseno de 30 é: 0.8660254037844387
O seno de 30 é: 0.5773502691896257
O valor da hipotenusa é: 11.180339887498949

```
vermelho = '\033[31m'  
verde = '\033[32m'  
azul = '\033[34m'  
ciano = '\033[36m'  
magenta = '\033[35m'  
amarelo = '\033[33m'  
preto = '\033[30m'  
branco = '\033[37m'  
restaura cor original = '\033[0;0m'  
negrito = '\033[1m'  
reverso = '\033[2m'  
fundo preto = '\033[40m'  
fundo vermelho = '\033[41m'  
fundo verde = '\033[42m'  
fundo azul = '\033[43m'  
fundo magenta = '\033[45m'  
fundo ciano = '\033[46m'  
fundo branco = '\033[47m'  
  
1 # Curso Técnico de Informática  
2 # Autor: Sebastião Marcos  
3 # Data: 26/10/2021  
4  
5 # Importando as bibliotecas  
6 import os  
7  
8 # Limpando o terminal  
9 os.system('cls')  
10  
11 print('*'*70)  
12 print('Cores no terminal do Python')  
13 print('*'*70)  
14  
15 # Vermelho  
16 print ('\033[31m'+ 'Saída em Vermelho' + '\033[0;0m')  
17 print()  
18  
19 # Verde  
20 print ('\033[32m'+ 'Saída em Verde' + '\033[0;0m')  
21 print()  
22  
23 # Verde  
24 print ('\033[34m'+ 'Saída em Azul' + '\033[0;0m')  
25 print()  
26  
27 # Amarelo com fundo verde  
28 print ('\033[42m'+ '\033[1m'+ '\033[33m'+ 'Isto eh amarelo negrito com fundo verde' + '\033[0;0m')  
29  
30 print('*'*70)  
31
```

- a. Faça um programa que receba um valor e mostre sua raiz quadrada. Para raízes que não são exatas, arredondar para cima ou para baixo. Faça a validação para números negativos avisando que o resultado será um número complexo.
- b. Faça um programa que receba 2 valores, faça a divisão entre eles, se a divisão não for inteira o programa deverá arredondar o resultado para cima e para baixo. Fazer a validação de divisão por 0.
- c. Faça um programa que receba as informações de base e expoente. Calcule a potência.
- d. Faça um programa que receba um ângulo qualquer. Em seguida, calcule o seno, cosseno e tangente do ângulo. Limite a saída com 2 casas decimais.
- e. Faça um programa para sortear um número de 1 a 20.
- f. Faça um programa onde o usuário tenta adivinhar o número que o computador está 'pensando'.
- g. Faça um programa que peça os valores de a, b e c de uma equação do 2º. Calcule as raízes das equação do 2º grau seguindo a fórmula: $\Delta = b^2 - 4ac$ $x = -b \pm \sqrt{\Delta} / 2a$

Strings são tipos de dados utilizados para representar cadeias de caracteres. Não só o Python, mas a maioria das linguagens possui esse tipo de dado e, geralmente, é representado por aspas simples (' texto ') ou aspas duplas (" texto "). No Python as cadeias de caracteres são 'quebradas' e a cada parte é anexada um índice. Assim é possível fazer uma varredura nessa cadeia de caracteres.

Declaração de uma String

```
frase = 'String em Python!'
texto = 'String em Python!'
```

Variável String

Alocação na memória

String	S	t	r	i	n	g		e	m		P	y	t	h	o	n	!
índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 frase = 'String em Python!'
12 texto = 'String em Python!'
13
14 print(f'Imprimindo a variável "frase": {frase}')
15 print(f'Imprimindo a variável "texto": {texto}'')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Imprimindo a variável "frase": String em Python!
Imprimindo a variável "texto": String em Python!

String	S	t	r	i	n	g		e	m		P	y	t	h	o	n	!
índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

frase [0] = S frase [10] = P

frase [0 : 10] =

S	t	r	i	n	g		e	m	
0	1	2	3	4	5	6	7	8	9

frase [0 : 10 : 2] =

S		n		m
0	2	4	6	8

Intervalo

Fim

Início

```

11 frase = 'String em Python!'
12 texto = 'String em Python!'
13
14 char_posicao_0 = frase[0]
15 char_posicao_10 = texto[10]
16 chars_0_a_10 = frase[0:10]
17 chars_0_a_10_com_intervalo_2 = texto[0:10:2]
18
19
20 print(f'Imprimindo o char na posição 0: {char_posicao_0}')
21 print(f'Imprimindo o char na posição 10: {char_posicao_10}')
22 print(f'Imprimindo os chars de 0 a 10: {chars_0_a_10}')
23 print(f'Imprimindo os chars de 0 a 10 com intervalo 2: {chars_0_a_10_com_intervalo_2}')

```

Fatiamento Strings

Agora vamos manipular essas cadeias de caracteres! O exemplo a seguir mostra como quebramos e acessamos as partes da nossa String. Importante: no Python uma letra maiúscula é diferente de uma minúscula como estudado anteriormente (case sensitive).

Imprimindo o char na posição 0: S
 Imprimindo o char na posição 10: P
 Imprimindo os chars de 0 a 10: String em
 Imprimindo os chars de 0 a 10 com intervalo 2: Srn m

Fatiamento Strings

Omitindo valores do fatiamento

String	S	t	r	i	n	g		e	m		P	y	t	h	o	n	!
índice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

frase [:] =

S	t	r	i	n	g		e	m	
0	1	2	3	4	5	6	7	8	9

frase [:] =

S	t	r	i	n	g		e	m		P	y	t	h	o	n	!
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Funções String

A linguagem de programação python traz várias funções de string prontas para utilizarmos. Todas as variáveis strings vêm com essas funcionalidades, basta chamar a função diretamente da variável que você deseja.

Função len():

Este método retorna o comprimento de nossa String, ou seja, a quantidade de caracteres que existe em nossa variável.

Função count():

Este método retorna quantas vezes um determinado caractere aparece em nossa String. Podemos usar também o fatiamento.

Função find():

Este método procura e retorna onde começa o caractere ou String. Quando o find() é usado com uma String que não existe, ele retorna -1.

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 print('Trabalhando com Strings: Funções')
12 print('='*70)
13
14 # Declaração
15 variavel_str = 'Estou estudando Python!'
16
17 quantidade_caracteres = len(variavel_str)
18 ocorrencia_caraceres = variavel_str.count('s', 0, 10)
19 posicao_caractere = variavel_str.find('Python!')
20
21 print(f'String: {variavel_str}')
22 print(f'Quantidade de caracteres: {quantidade_caracteres}')
23 print(f'Ocorrência de "s" na String: {ocorrencia_caraceres} ocorrências')
24 print(f'Índice da Palavra "Python!": Posição {posicao_caractere}')
25 print('='*70)
```

```

Trabalhando com Strings: Funções
=====
String: Estou estudando Python!
Quantidade de caracteres: 23
Ocorrência de "s" na String: 2 ocorrências
Índice da Palavra "Python!": Posição 16
-----
```

Método upper():

Este método altera minúsculas para maiúsculas.

Função lower():

Este método altera maiúsculas para minúsculas.

Função capitalize():

Este método altera a primeira letra da frase para maiúscula.

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 26/10/2021
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 print('Trabalhando com Strings: Funções')
12 print('*'*70)
13
14 # Declaração
15 variavel_str = 'Estou estudando Python!'
16
17 variavel_maiuscula = variavel_str.upper()
18 variavel_minuscula = variavel_str.lower()
19 variavel_capitalizada = variavel_str.capitalize()
20
21 print(f'String: {variavel_str}')
22 print(f'Toda String maiúscula: {variavel_maiuscula}')
23 print(f'Toda String minúscula: {variavel_minuscula}')
24 print(f'Toda String capitalizada: {variavel_capitalizada}')
25 print('*'*70)
```

Trabalhando com Strings: Funções

String: Estou estudando Python!

Toda String maiúscula: ESTOU ESTUDANDO PYTHON!

Toda String minúscula: estou estudando python!

Toda String capitalizada: Estou estudando python!

Função split():

Divide uma String em outras Strings. O espaço é considerado a área divisora.

Função Join()

O join()método pega todos os itens em um iterável e os une em uma string.

Uma string deve ser especificada como o separador.

Função strip():

Remove espaços no começo e no final da String.

Também pode ser usada desta forma:

rstrip(): Apaga caracteres vazios à direita

lstrip(): Apaga caracteres vazios à esquerda

```

11  print('Trabalhando com Strings: Funções')
12  print('*'*80)
13
14  # Declaração
15  variavel_str = 'Estou estudando Python!'
16  lista = ['Olá', 'Mundo!']
17
18  separando_caracteres = variavel_str.split()
19  juntar_string = ' '.join(lista)
20  frase_com_espaços = ' Olá mundo! '
21  frase_sem_espaços = frase_com_espaços.strip()
22
23  print(f'String: {variavel_str}')
24  print(f'Palavras separadas, "Gera uma [lista]": {separando_caracteres}')
25  print(f'Antes: {lista} -- Depois: {juntar_string}')
26  print('*'*80)
27  print(f'{frase_com_espaços}')
28  print(f'Quantidade de carateres na frase com espaços: {len(frase_com_espaços)}')
29  print(f'Quantidade de carateres na frase sem espaços: {len(frase_sem_espaços)}')
30  print('*'*80)

```

Trabalhando com Strings: Funções

=====

String: Estou estudando Python!

Palavras separadas, "Gera uma [lista)": ['Estou', 'estudando', 'Python!']

Antes: ['Olá', 'Mundo!'] -- Depois: Olá Mundo!

Olá mundo!

Quantidade de carateres na frase com espaços: 16

Quantidade de carateres na frase sem espaços: 10

Função replace():

Esta função substitui uma parte do texto por uma outra String. A palavra replace significa substituir.

Operador in:

Ele faz uma procura para ver se uma String existe dentro da outra.

```
11  print('Trabalhando com Strings: Funções')
12  print('='*80)
13
14 # Declaração
15 variavel_str = 'Estou estudando Python!'
16
17 resultado_do_replace = variavel_str.replace('Python', 'Java')
18
19 if ('estudando' in variavel_str):
20     resposta = 'Verdadeiro'
21 else:
22     resposta = 'Falso'
23
24 print(f'String original: {variavel_str}')
25 print(f'Quantidade de caracteres: {resultado_do_replace}')
26 print(f'Existe a palavra "estudando" na string "{variavel_str}": {resposta}')
27 print('-'*80)
```

```
Trabalhando com Strings: Funções
=====
String original: Estou estudando Python!
Quantidade de caracteres: Estou estudando Java!
Existe a palavra "estudando" na string "Estou estudando Python!": Verdadeiro
```

- a. Faça um programa que peça separadamente o nome, o nome do meio e o sobrenome de uma pessoa. Depois imprima o nome completo.
- b. Faça um programa que receba o nome 'João da Silva', e em seguida troque o 'Silva' para 'Oliveira'.
- c. Faça um programa que leia o nome de uma pessoa e verifique se há a palavra 'Oliveira' neste nome, mostrando True ou False.
- d. Faça um programa que leia uma frase e depois mostre na tela:
- A frase em minúsculo, a frase em maiúscula, qtde de caracteres na frase, quantas letras tem a 2^a palavra na frase.
- e. Faça um programa que receba uma frase, depois mostre quantas vezes as vogais foram utilizadas.
- f. Faça um programa que receba o nome completo de uma pessoa e depois imprima esse nome separadamente.
- g. Faça um programa que receba um número milhar e mostre na tela:
- Quantidade de unidades, quantidade de dezenas, quantidade de centenas; quantidade de milhar.
- h. Faça um programa que leia o nome de um aluno e mostre quantas vezes aparece a letra 'o' e em que posição ela aparece a 1^a e pela última vez.
- i. Faça um programa que receba o nome completo de uma pessoa, em seguida mostre o primeiro e o último nome.

Observação

É obrigatório a saída formatada!

O código precisa ter o cabeçalho padrão

O código precisa ser comentado

Implemente validação quando achar necessário

Enviar somente os exercícios indicados com a cor laranja.

Assim como em outras linguagens de programação, as estruturas de repetição são utilizadas quando queremos que um bloco de código seja executado um número qualquer de vezes.

Comando For()

O comando for é usado quando se quer iterar sobre um bloco de código um número determinado de vezes. A essas iterações podem ser agregadas com condicionais e interrupções do laço. Esse conceito também pode ser aplicado ao while

Comando While()

O comando while é usado quando queremos que o bloco de código seja repetido até que uma condição seja satisfeita, ou seja, é preciso que uma expressão lógica seja verdadeira. Assim que ela se tornar falsa, a estrutura While é interrompida.

Quando usar o For e quando usar o While?

For: Quando sabemos o número de vezes que queremos executar

While: Quando não sabemos o número de vezes

```

12 print('-' * 70)
13 print('ESTRUTURA DE CONTROLE FOR')
14 print('=' * 70)
15
16 inicio = 0 # só para exemplificar
17 fim = 10 # só para exemplificar
18 passo = 1 # só para exemplificar
19
20 print()
21
22 for var_iteradora in range(inicio, fim, passo):
23     print(var_iteradora, end=' ')
24
25 print()
26 print('=' * 70)
27
28 print()
29 print()
30 print()
31
32
33 print('ESTRUTURA DE CONTROLE WHILE')
34 print('=' * 70)
35
36 # Flag
37 contador = 0
38
39 while (contador <= 10):
40     print(contador, end=' ')
41     contador += 1
42
43 print()
44 print('=' * 70)
```

ESTRUTURA DE CONTROLE FOR
=====

0 1 2 3 4 5 6 7 8 9

ESTRUTURA DE CONTROLE WHILE
=====

0 1 2 3 4 5 6 7 8 9 10
=====

Comando For... range

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 22/11/2022
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # Título
12 print('-' * 70)
13 print('ESTRUTURA DE CONTROLE FOR RANGE')
14 print('=' * 70)
15
16 print()
17
18 for var_iteradora in range(1, 7):
19     # end= coloca os números em uma mesma linha
20     print(f'Valor: {var_iteradora}', end=" | ")
21
22 print()
23 print()

```

ESTRUTURA DE CONTROLE FOR RANGE
=====

Valor: 1 | Valor: 2 | Valor: 3 | Valor: 4 | Valor: 5 | Valor: 6 |

Comando For... range...input

```

1 #Curso Técnico de Informática
2 #Autor: Sebastião Marcos
3 #Data início: 16/11/2021
4 #Término: 16/11/2021
5 #Estudo de Strings
6
7 import os
8
9 # Limpando o terminal
10 os.system('cls')
11
12 # Título
13 print('-' * 70)
14 print('ESTRUTURA DE CONTROLE FOR RANGE')
15 print('=' * 70)
16
17 print()
18
19 for var_iteradora in range(1, 5):
20     cor = str(input(f'Digite a {var_iteradora}ª cor: '))
21
22 print()
23 print() -----  
ESTRUTURA DE CONTROLE FOR RANGE  
=====
```

Digite a 1ª cor: Verde
 Digite a 2ª cor: Vermelho
 Digite a 3ª cor: Amarelo
 Digite a 4ª cor: Azul

For...range... somando números

```

11 # Título
12 print('-' * 50)
13 print('ESTRUTURA DE CONTROLE SOMATÓRIO')
14 print('=' * 50)
15
16 print()
17
18 # Declaração
19 soma = 0
20
21 for var_iteradora in range(0, 4):
22     numero = int(input(f'Digite o {var_iteradora + 1}º número: '))
23
24     # Cálculo
25     soma += numero # mesma coisa: soma = soma + numero
26
27 print('-' * 50)
28 print(f'A soma dos números é: {soma}')
29 print('=' * 50)
30 print()

```

ESTRUTURA DE CONTROLE SOMATÓRIO
=====

```

Digite o 1º número: 5
Digite o 2º número: 5
Digite o 3º número: 5
Digite o 4º número: 5
=====
A soma dos números é: 20
=====
```

Comando For... range...input...if else

```

11 # Título
12 print('-' * 50)
13 print('ESTRUTURA DE CONTROLE COM INPUT E IF')
14 print('=' * 50)
15
16 print()
17
18 # Declaração
19 soma = 0
20
21 for var_iteradora in range(0, 4):
22     numero = int(input(f'{var_iteradora + 1}º número: '))
23
24     # Condisional
25     if (numero % 2 == 0):
26         print(f'O número {numero} é par')
27     else:
28         print(f'O número {numero} é ímpar')
29
30 print('=' * 50) ----- ESTRUTURA DE CONTROLE COM INPUT E IF
31 print()

```

```

1º número: 10
O número 10 é par
2º número: 11
O número 11 é ímpar
3º número: 10
O número 10 é par
4º número: 5
O número 5 é ímpar
=====
```

```

11 # Título
12 print('-' * 50)
13 print('ESTRUTURA DE CONTROLE VALIDAÇÃO E CASTING')
14 print('=' * 50)
15
16 print()
17
18 # Declaração
19 soma = 0
20
21 # Estrutura de Controle
22 for c in range(0, 5):
23
24     # Entrada de Dados
25     numero = input('Digite um número [0-5]: ')
26
27     # Validação
28     if (not (numero.isnumeric())):
29         print(f'{numero}' " não é um número!")
30         print()
31     else:
32         # Casting da variável, ou seja, vai virar um inteiro
33         numero = int(numero)
34
35         # Validando o intervalo pedido
36         if (numero >= 0 and numero <= 5):
37             print(f'O número {numero} é válido!')
38             print()
39         else:
40             print(f'O número {numero} não é válido!')
41             print()
42
43     print('-' * 70)
44     print()

```

For...range... validação e casting

ESTRUTURA DE CONTROLE VALIDAÇÃO E CASTING

=====

Digite um número [0-5]: a
"a" Entrada inválida!

Digite um número [0-5]: 1
O número 1 está dentro do intervalo [0-5]!

Digite um número [0-5]: 6
"6" Entrada inválida!

Digite um número [0-5]: -1
"-1" Entrada inválida!

Digite um número [0-5]: 3
O número 3 está dentro do intervalo [0-5]!

For...range... break

A instrução break interrompe o ciclo de repetição na qual está contida. Essa instrução pode ser usada dentro de uma condicional if.

```
11 # Título
12 print('-' * 70)
13 print('ESTRUTURA DE CONTROLE: BREAK')
14 print('=' * 70)
15
16 print()
17
18 for c in range(0, 10):
19
20     print(f'Valor: {c}')
21
22     # Condição para terminar a contagem
23     if (c == 5):
24         print(f'Contagem interropida no {c}')
25         break
26
27 print('-' * 70)
28 print('Fim do programa!')
29 print()
```

ESTRUTURA DE CONTROLE: BREAK

```
=====
Valor: 0
Valor: 1
Valor: 2
Valor: 3
Valor: 4
Valor: 5
Contagem interropida no 5
=====
Fim do programa!
```

For...range... continue

A instrução continue salta o ciclo de repetição na qual está contida. Essa instrução pode ser usada dentro de uma condicional if.

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 22/11/2022
4
5 # Importando as bibliotecas
6 import os
7
8 # Limpando o terminal
9 os.system('cls')
10
11 # Título
12 print('-' * 70)
13 print('ESTRUTURA DE CONTROLE: CONTINUE')
14 print('=' * 70)
15 print()
16
17 for c in range(1, 11):
18     if c == 5:
19         print(f'O número {c} está fora do loop')
20         continue    # Salta e o ciclo continua
21
22     print('o número é ' + str(c))
23
24 print('-' * 70)
25 print()
```

ESTRUTURA DE CONTROLE: CONTINUE

```
o número é 1
o número é 2
o número é 3
o número é 4
O número 5 está fora do loop
o número é 6
o número é 7
o número é 8
o número é 9
o número é 10
```

While...else

Como vimos no começo do estudo de Estruturas de Controle, o comando while é usado quando queremos que o bloco de código seja repetido até que uma condição seja satisfeita, ou seja, é preciso que uma expressão lógica dada seja verdadeira. Assim que ela se tornar falsa, a estrutura será interrompida.

```

1  # Curso Técnico de Informática
2  # Autor: Sebastião Marcos
3  # Data: 22/11/2022
4
5  # Importando as bibliotecas
6  import os
7
8  # Limpando o terminal
9  os.system('cls')
10
11 # Título
12 print('-' * 70)
13 print('ESTRUTURA DE CONTROLE WHILE ELSE')
14 print('=' * 70)
15 print()
16
17 print()
18
19 # Declaração
20 opcao = 'S' # Flag
21
22 while (opcao != 's'):
23     opcao = str(input('Digite um nome: (S - sair) ')).lower()
24 else:
25     print('Você digitou "s", fim do programa!')
26
27 print('-' * 70)
28 print()
```

ESTRUTURA DE CONTROLE WHILE ELSE

```

Digite um nome: (S - sair) a
Digite um nome: (S - sair) b
Digite um nome: (S - sair) sim
Digite um nome: (S - sair) 2
Digite um nome: (S - sair) s
Você digitou "s", fim do programa!
```

While...else

Vamos usar o mesmo exemplo, agora utilizando os comandos if...else...break:

```
11 # Título
12 print('*'*70)
13 print('ESTRUTURA DE CONTROLE WHILE ELSE')
14 print('=' * 70)
15 print()
16
17 # Estrutura de Controle
18 while (True):
19
20     # Entrada
21     nome = str(input('Digite um nome [s - Sair]: ')).lower()
22
23     # Condicional
24     if (nome != 's'):
25         print('Continue digitando...')
26     else:
27         print('*'*70)
28         print('Você digitou "s" para sair!')
29
30     # interrompe o laço
31     break
32
33 print('*'*70)
34 print()
```

ESTRUTURA DE CONTROLE WHILE ELSE

```
=====
Digite um nome [s - Sair]: a
Continue digitando...
Digite um nome [s - Sair]: -1
Continue digitando...
Digite um nome [s - Sair]: sim
Continue digitando...
Digite um nome [s - Sair]: s
=====
Você digitou "s" para sair!
```

While...continue

A instrução continue interrompe a execução de apenas 1 ciclo, mas não termina a execução do laço. Exemplo para verificar o número par. Se for par, ele salta o número:

ESTRUTURA DE CONTROLE WHILE ELSE CONTINUE
=====

```
Contador = 1
Contador = 3
Contador = 5
Contador = 7
Contador = 9
Contador = 11
Bloco do while...else 11
-----
Fim do programa!
```

```
11 # Título
12 print('-' * 70)
13 print('ESTRUTURA DE CONTROLE WHILE ELSE CONTINUE')
14 print('=' * 70)
15 print()
16
17 contador = 0 # Flag
18
19 while (contador <= 10):
20
21     # Soma o contador
22     contador += 1 # é o mesmo que contador = Contador + 1
23
24     # Condicional
25     if (contador % 2 == 0):
26         continue
27     # Imprime o contador
28     print(f'Contador = {contador}')
29 else:
30     print(f'Bloco do while...else {contador}')
31
32 print('-' * 70)
33 print('Fim do programa!')
34 print()
```

- a. Faça um programa que imprima os números no intervalo entre 1 e 100.
- b. Evolua o programa anterior para um código que pergunte ao usuário qual o intervalo que ele deseja ver impresso.
- c. Faça um programa que imprima os números no intervalo entre 1 e 10 em ordem inversa.
- d. Faça um programa que imprima os números pares entre 0 e 100.
- e. Faça um programa que some a quantidade de números pares encontrados no intervalo entre 0 e 100.
- f. Faça um programa que imprima os números primos entre 0 e 100.
- g. Faça um programa que calcule os números primos em um intervalo pré-determinado pelo usuário.
- h. Faça um programa que imprima os valores no intervalo definidos pelo usuário. Defina 3 números que deverão ser ignorados, ou seja, que não serão impressos na tela.
- i. Faça um algoritmo que imprima a frase "estou em looping" e, em seguida, solicite ao usuário digitar uma letra. Caso a letra seja o "f" finalize a aplicação. Do contrário, imprima novamente a mesma frase até que o caractere "f" seja digitado.

- j. Crie um programa que realiza a contagem de 1 até 100, usando apenas de números ímpares, ao final do processo exiba em tela quantos números ímpares foram encontrados nesse intervalo, assim como a soma dos mesmos.
- k. Crie um programa que pede que o usuário digite um nome ou uma frase, verifique se esse conteúdo digitado é um palíndromo ou não, exibindo em tela esse resultado.
- l. Faça um programa que peça o login e senha de usuário cadastrado em um sistema. Se o usuário errar o login, o sistema irá responder que o login está inválido e retornará para o pedido de entrada.

Listas

São estruturas heterogêneas de dados que armazenam uma coleção ordenada de itens em Python.

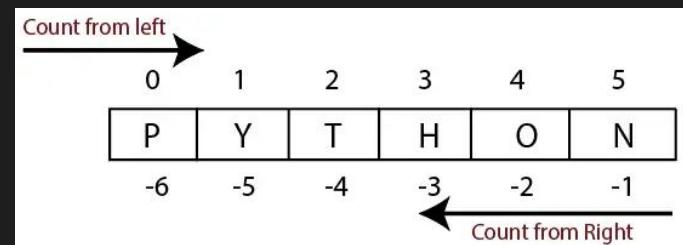
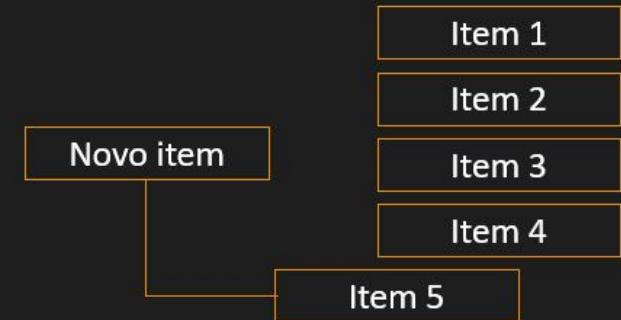
A lista geralmente é representada por colchetes [].

Sintaxe:

```
lista1 = [1, 2, 3],  
lista2 = ['a', 'e', 'i'],  
lista3 = [1.1, 'John', 80],  
lista4 = [['a', 'e'], [1, 2, 3]]
```

Características:

- Listas são mutáveis e são ordenadas por ordem de entradas
- Itens novos são adicionados ao final da lista
- O primeiro item adicionado sempre será o primeiro da lista



```

1 #Curso Técnico de Informática
2 #Autor: Sebastião Marcos
3 #Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 # Declaração
15 lista_numeros_inteiros = [1, 2, 3, 4]
16 lista_vogais = ['a', 'e', 'i', 'o', 'u']
17 lista_nomes = ['John', 'Jane', 'Carol']
18 lista_heterogenea = ['John', 80, 1.90, 'AB']
19 lista_dentro_lista = [[1, 2, 3, 4], ['a', 'e', 'i', 'o', 'u']]
20
21 # Imprimindo as listas
22 print(f'Lista de números: {lista_numeros_inteiros}')
23 print(f'Lista de vogais: {lista_vogais}')
24 print(f'Lista de nomes: {lista_nomes}')
25 print(f'Lista misturada: {lista_heterogenea}')
26 print(f'Lista de lista: {lista_dentro_lista}')
27
28 # Varrendo os índices manualmente
29 lista_num_indice_0 = lista_numeros_inteiros[0]
30 lista_vogais_indice_1 = lista_vogais[1]
31 lista_nomes_indice_2 = lista_nomes[2]
32 lista_heterogenea_indice_3 = lista_heterogenea[3]
33 lista_num_indice_1 = lista_dentro_lista[1]
34
35 print('*' * 70)
36 print('Acessando os elementos de uma lista')
37 print('*' * 70)
38 print(f'Lista de números: {lista_num_indice_0}')
39 print(f'Lista de vogais: {lista_vogais_indice_1}')
40 print(f'Lista de nomes: {lista_nomes_indice_2}')
41 print(f'Lista heterogênea: {lista_heterogenea_indice_3}')
42 print(f'Lista de lista: {lista_num_indice_1}')
43
44 print('*' * 70)
45 print('Fim do programa!')
46 print('*' * 70)

```

```

Lista de números: [1, 2, 3, 4]
Lista de vogais: ['a', 'e', 'i', 'o', 'u']
Lista de nomes: ['John', 'Jane', 'Carol']
Lista misturada: ['John', 80, 1.9, 'AB']
Lista de lista: [[1, 2, 3, 4], ['a', 'e', 'i', 'o', 'u']]
-----
Acessando os elementos de uma lista
=====
Lista de números: 1
Lista de vogais: e
Lista de nomes: Carol
Lista heterogênea: AB
Lista de lista: ['a', 'e', 'i', 'o', 'u']
-----
Fim do programa!
-----
```

Fatiando uma lista

Para executar o fatiamento de uma lista, basta seguir os mesmo passos do fatiamento de String em Python. Utilizamos colchetes após a variável e em seguida declaramos o início, fim e intervalo caso seja necessário.

```

8  os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 # Fatiamento de Listas
15 lista_mista = ['a', 'b', 3, 'John', 'e', 500, 'g', 'h']
16
17 # Fatia o 1º elemento
18 lista_fatiada1 = lista_mista[0]
19 # Fatia todos os elementos
20 lista_fatiada2 = lista_mista[0:]
21 # Fatia os elementos do índice 0 até o (índice 6 - 1)
22 lista_fatiada3 = lista_mista[0:6]
23 # Fatia os elementos do índice 0 até o índice 6 de 2 em 2
24 lista_fatiada4 = lista_mista[0:6:2]
25 # Fatia os elementos da direita para esquerda
26 lista_fatiada5 = lista_mista[::-1]
27
28 print(f'Fatiando uma Lista: {lista_fatiada1}\n')
29 print(f'Fatiando uma Lista: {lista_fatiada2}\n')
30 print(f'Fatiando uma Lista: {lista_fatiada3}\n')
31 print(f'Fatiando uma Lista: {lista_fatiada4}\n')
32 print(f'Fatiando uma Lista: {lista_fatiada5}\n')
33
34 print('*'*70)
35 print('Fim do programa!')
36 print('*'*70)
```

ESTRUTURA DE DADOS: LISTAS []
=====

Fatiando uma Lista: a

Fatiando uma Lista: ['a', 'b', 3, 'John', 'e', 500, 'g', 'h']

Fatiando uma Lista: ['a', 'b', 3, 'John', 'e', 500]

Fatiando uma Lista: ['a', 3, 'e']

Fatiando uma Lista: ['h', 'g', 500, 'e', 'John', 3, 'b', 'a']

Fim do programa!

Métodos / Funções para Listas

`append()`: Adiciona um elemento ao final da lista.

`copy()`: Retorna uma cópia superficial da lista (import copy).

`pop()`: Elimina e retorna um elemento da lista pelo índice.

`reverse()`: Inverte a ordem dos elementos da lista.

`sort(lista, reverse=True)`: Classifica os elementos de uma lista.

`sorted(lista, reverse=True)`: Classifica os elementos de qualquer iterável.

`insert(i, v)`: Insere um elemento na lista utilizando um índice

`remove()`: Elimina um elemento da lista através do valor.

ESTRUTURA DE DADOS: LISTAS []
=====

Lista de números: [3, 5, 1, 4, 2, 6, 100]

Lista nova: [3, 5, 1, 4, 2, 6, 100]

Lista nova, valor deletado: 100

Lista nova: [3, 5, 1, 4, 2, 6]

Lista ordenada: [1, 2, 3, 4, 5, 6]

Lista invertida: [6, 5, 4, 3, 2, 1]

Elemento 6 removido: [5, 4, 3, 2, 1]

Fim do programa!

Acrescentar o
insert()

```

11 print('-' * 70)
12 print('ESTRUTURA DE DADOS: LISTAS [ ]')
13 print('=' * 70)
14
15 # Declaração
16 lista_numeros_inteiros = [3, 5, 1, 4, 2, 6]
17
18 # Método append()
19 lista_numeros_inteiros.append(100)
20
21 # Método copy()
22 lista_nova = copy.copy(lista_numeros_inteiros)
23 print(f'Lista de números: {lista_numeros_inteiros}')
24 print(f'Lista nova: {lista_nova}\n')
25
26 # Método pop()
27 valor_apagado = lista_nova.pop(6) # apaga pelo índice
28 print(f'Lista nova, valor deletado: {valor_apagado}')
29 print(f'Lista nova: {lista_nova}\n')
30
31 # Método sorted()
32 # listaOrdenada = sorted(listaNova, reverse=True) Ordem decrescente
33 lista_ordenada = sorted(lista_nova)
34 print(f'Lista ordenada: {lista_ordenada}\n')
35
36 # Método reverse()
37 lista_ordenada.reverse()
38 print(f'Lista invertida: {lista_ordenada}\n')
39
40 # Método remove()
41 lista_ordenada.remove(6)
42 print(f'Elemento 6 removido: {lista_ordenada}\n')
43
44 print('*'*70)
45 print('Fim do programa!')
46 print('*'*70)

```

Métodos / Funções para Listas

`extend()`: Insere mais de um elemento em uma lista.

`index()`: O método retorna a primeira ocorrência do valor especificado.

`max()`: Essa função retornará o valor mais alto dos valores inseridos.

`min()`: Essa função retornará o valor mais baixo dos valores inseridos.

`clear()`: apaga todos os elementos de uma lista.

`len()`: Essa função mostra o número de elementos em uma lista.

```

10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 # Método sort()
15 lista_precos = [250.50, 90.30, 500.60]
16 print(f'Antes: {lista_precos}', )
17 # ListaPrecos.sort(reverse = True) ordem descendente
18 lista_precos.sort()
19 print(f'Depois: {lista_precos}', )
20 print('='*70)
21

```

```

22 # extend():
23 a = [1, 2, 3]
24 print(f'Antes: {a}')
25 a.extend([4, 5])
26 print(f'Itens adicionados: {a}')
27 print('='*70)
28
29 # index()
30 meses = ['Janeiro', 'Fevereiro', 'Março']
31 indice_mes = meses.index('Fevereiro')
32 print(f'O mês está no índice: {indice_mes}')
33 print('='*70)
34
35 # max() e min()
36 valores = [1, 12.5, 8]
37 maior_valor = max(valores)
38 print(f'Lista: {valores}')
39 print(f'O maior valor: {maior_valor}')
40 print('='*70)
41

```

```
42 # clear()
43 listaNomes = ['John', 'Jane', 'Carol']
44 print(f'Antes: {listaNomes}', )
45 listaNomes.clear()
46 print(f'Depois: {listaNomes}', )
47 print('*'*70)
48
49 # len()
50 lista = [1, 2, 3, 'a', 'b', 'c', 5.5]
51 tamanhoLista = len(lista)
52
53 # insert()
54 listaNumeros = [200, 300, 500]
55 print(f'Antes: {listaNumeros}')
56 listaNumeros.insert(2, 400)
57 print(f'Depois: {listaNumeros}')
58
59 print('*'*70)
60 print('Fim do programa!')
61 print('*'*70)
```

ESTRUTURA DE DADOS: LISTAS []
=====

Antes: [250.5, 90.3, 500.6]
Depois: [90.3, 250.5, 500.6]

Antes: [1, 2, 3]
Itens adicionados: [1, 2, 3, 4, 5]

0 mês está no índice: 1

Lista: [1, 12.5, 8]
0 maior valor: 12.5

Antes: ['John', 'Jane', 'Carol']
Depois: []

Antes: [200, 300, 500]
Depois: [200, 300, 400, 500]

Fim do programa!

Entrada de dados com for e input em listas

Podemos utilizar entrada de dados para inserir valores em uma lista. Esse procedimento fica mais dinâmico quando utilizamos laços de repetição.

ESTRUTURA DE DADOS: LISTAS []

```
Entre com o nome do aluno: John Doe
Entre com o nome do aluno: Jane Doe
Entre com o nome do aluno: Suzan
Entre com o nome do aluno: Elizabeth
Entre com o nome do aluno: Maria
Impressão dos nomes de alunos:
['John Doe', 'Jane Doe', 'Suzan', 'Elizabeth', 'Maria']
```

Fim do programa!

```
1 #Curso Técnico de Informática
2 #Autor: Sebastião Marcos
3 #Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 # Criando uma lista
15 lista_alunos = []
16
17 for c in range(0, 5):
18     nome = str(input('Entre com o nome do aluno: '))
19
20     # Guardando em uma lista
21     lista_alunos.append(nome)
22
23 print('Impressão dos nomes de alunos:')
24 print(f'{lista_alunos}\n')
25 print('*'*70)
26 print('Fim do programa!')
27 print('*'*70)
```

Entrada de dados com for e input em listas

Podemos utilizar entrada de dados para inserir valores em uma lista. Esse procedimento fica mais dinâmico quando utilizamos laços de repetição.

```
ESTRUTURA DE DADOS: LISTAS [ ]
```

```
=====
```

```
Entre com o nome do aluno: Jane Austen
Entre com o nome do aluno: Maria Firmina dos Reis
Entre com o nome do aluno: Virginia Wolf
Entre com o nome do aluno: Clarice Lispector
Entre com o nome do aluno: Chimamanda Ngozi
```

```
Impressão dos nomes de alunos:
```

```
Jane Austen Maria Firmina dos Reis Virginia Wolf Clarice Lispector Chimamanda
Ngozi
```

```
Fim do programa!
```

```
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 # Criando uma lista
15 lista_alunos = []
16
17 for c in range(0, 5):
18     nome = str(input('Entre com o nome do aluno: '))
19
20     # Guardando em uma lista
21     lista_alunos.append(nome)
22
23 print()
24 print('Impressão dos nomes de alunos:')
25
26 # utilizando o len() para saber a quantidade de alunos
27 for c in range(len(lista_alunos)):
28     print(lista_alunos[c], end=' ')
29     if c == 3:
30         print()
31
32 print()
33 print('*'*70)
34 print('Fim do programa!')
35 print('*'*70)
```

Iteração em listas: for in enumerate()

Em Python, um for geralmente é escrito como um loop sobre um objeto iterável. Isso significa que você não precisa de uma variável de contagem para acessar itens no iterável. Às vezes, porém, você deseja ter uma variável que muda a cada iteração do loop. Em vez de criar e incrementar você mesmo uma variável, você pode usar a função `enumerate()` para obter um contador e o valor do iterável ao mesmo tempo!

ESTRUTURA DE DADOS: LISTAS []

```
=====
Indice: 0 = Número: 1
Indice: 1 = Número: 2
Indice: 2 = Número: 3
Indice: 3 = Número: 4
Indice: 4 = Número: 5
Indice: 5 = Número: 6
Indice: 6 = Número: 7
Indice: 7 = Número: 8
Indice: 8 = Número: 9
Indice: 9 = Número: 10
=====
```

55

Fim do programa!

```
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 soma = 0
15
16 # Criando uma lista
17 lista_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
18
19 # percorrendo a lista com o enumerate()
20 # O comando enumerate adiciona um índice
21 # para cada valor de nossa lista
22 # start opcional, para não começar no índice 0
23 # enumerate(listaNumeros, start = 1)
24
25 for indice, numeros in enumerate(lista_numeros):
26     soma += numeros
27     # imprimindo os valores
28     print(f'Indice: {indice} = Número: {numeros}\n')
29
30 print('-'*70)
31 print(soma)
32 print('Fim do programa!')
33 print('=' * 70)
```

Iteração em listas: `for in enumerate()`

O operador `in` verifica se o operando à sua esquerda, está contido na lista à sua direita, da mesma forma que o operador `not in` que verifica o contrário.

Estes são 2 operadores nativos para verificar se um determinado objeto está contido em uma lista. A palavra `in`, do Inglês, significa "contido em". Essa é uma maneira para simplificar a verificação se o elemento X está contido na lista Y.

ESTRUTURA DE DADOS: LISTAS []

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

0 número 3 está na posição 2

['John', 'Jane', 'Carol']

O nome Maria foi acrescentado

['John', 'Jane', 'Carol', 'Maria']

Fim do programa!

```

10 print('-'*70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 # Criando uma lista
15 lista_numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
16
17 if (3 in lista_numeros):
18     print(lista_numeros)
19     posicao = lista_numeros.index(3)
20     print(f'O número 3 está na posição {posicao}')
21 else:
22     print('O elemento não consta na listagem')
23
24 print()
25
26 lista_nomes = ['John', 'Jane', 'Carol']
27
28 if ('Maria' not in lista_nomes):
29     # Antes
30     print(lista_nomes)
31
32     # Não está na lista, acrescentar
33     lista_nomes.append('Maria')
34
35     print('\nO nome Maria foi acrescentado')
36     print(lista_nomes)
37
38 print()
39 print('*'*70)
40 print('Fim do programa!')
41 print('*'*70)

```

Somando Valores em uma lista

Um exemplo para demonstrar como aplicar um cálculo matemático aos elementos de uma lista.

ESTRUTURA DE DADOS: LISTAS []

Entre com o 1º: 100
Entre com o 2º: 100
Entre com o 3º: 100
Entre com o 4º: 100
Entre com o 5º: 100

Impressão dos números inteiros
100 100 100 100 100

A soma dos valores é: 500
Fim do programa!

```
10  print('*' * 70)
11  print('ESTRUTURA DE DADOS: LISTAS [ ]')
12  print('=' * 70)
13
14  # Criando uma lista
15  lista_numeros = []
16  soma_valores = 0
17
18  for c in range(0, 5):
19      numero = int(input(f'Entre com o {c+1}º: '))
20
21      # Guardando em uma lista
22      lista_numeros.append(numero)
23      soma_valores += numero
24
25  print()
26  print('Impressão dos números inteiros')
27
28  # utilizando o len() para saber a quantidade de alunos
29  for c in range(len(lista_numeros)):
30      print(lista_numeros[c], end=' ')
31
32  print()
33  print('*' * 70)
34  print(f'A soma dos valores é: {soma_valores}')
35  print('Fim do programa!')
36  print('*' * 70)
```

Lista dentro de uma lista

```
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('*'*70)
11 print('ESTRUTURA DE DADOS: LISTAS [ ]')
12 print('=' * 70)
13
14 # X O X
15 # X X O
16 # O O O
17
18 # Similar a um array de 3 Dimensões
19 jogo_velha = [
20     |    | 0   1   2
21     |    | ['X', 'O', 'X'], # 0
22     |    | ['X', 'X', 'O'], # 1
23     |    | ['O', 'O', 'O'] # 2
24 ]
25
26 # pegando manualmente os valores
27 print(f'Linha 1 coluna 1 {jogo_velha[1][1]}')
28 print(f'Linha 0 coluna 2 {jogo_velha[0][2]}')
29
30 # varrendo com for range
31 for linha in range(0, len(jogo_velha)):
32     for coluna in range(0, len(jogo_velha)):
33         print(jogo_velha[linha][coluna], end=' ')
34     print()
```

```
ESTRUTURA DE DADOS: LISTAS [ ]
```

```
Linha 1 coluna 1 X
```

```
Linha 0 coluna 2 X
```

```
X O X
```

```
X X O
```

```
O O O
```

```
PS D:\Trabalho\Senac\Turma 0277\UC10\python>
```

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-'*70)
11 print('ESTRUTURA DE DADOS: COMPREENSÃO EM LISTAS [ ]')
12 print('=' * 70)
13
14 lista_numeros = [1, 2, 3, 4, 5]
15
16 # triplicar os valores
17 lista_nova_triplicada = []
18
19 for item in lista_numeros:
20     lista_nova_triplicada.append(item * 3)
21
22 print(f'Lista triplicada: {lista_nova_triplicada}')
23
24 # Compreensão
25 lista_nova_triplicada_2 = [item * 3 for item in lista_numeros]
26 print(f'Lista triplicada: {lista_nova_triplicada_2}\n')
```

List comprehension

Uma compreensão de lista é uma construção sintática disponível em algumas linguagens de programação para criação de uma lista baseada em listas existentes. Ela segue a forma da notação de [definição de conjunto](#) matemática como forma distinta para uso de funções de [mapa](#) e [filtro](#).

```
ESTRUTURA DE DADOS: COMPREENSÃO EM LISTAS [ ]
```

```
Lista triplicada: [3, 6, 9, 12, 15]
```

```
Lista triplicada: [3, 6, 9, 12, 15]
```

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('*' * 70)
11 print('ESTRUTURA DE DADOS: COMPREENSÃO EM LISTAS [ ]')
12 print('=' * 70)
13
14 lista_numeros = [100, 200, 300, 500, 600]
15
16 # triplicar os valores
17 lista_com_juros = []
18
19 for item in lista_numeros:
20     lista_com_juros.append(item + (item * .10))
21
22
23 print(f'Lista triplicada: {lista_com_juros}')
24
25 # Compreensão
26 lista_com_juros_2 = [item + (item * .10) for item in lista_numeros]
27 print(f'Lista triplicada: {lista_com_juros_2}\n')
```

```
ESTRUTURA DE DADOS: COMPREENSÃO EM LISTAS [ ]
=====
Lista triplicada: [110.0, 220.0, 330.0, 550.0, 660.0]
Lista triplicada: [110.0, 220.0, 330.0, 550.0, 660.0]
```

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('*' * 70)
11 print('ESTRUTURA DE DADOS: COMPREENSÃO EM LISTAS [ ]')
12 print('=' * 70)
13
14 lista_precos = [100, 200, 300, 500, 600]
15
16 # triplicar os valores
17 lista_com_juros = []
18
19 for valor in lista_precos:
20     if valor < 300:
21         lista_com_juros.append(valor + (valor * .10))
22
23 print(f'Lista triplicada: {lista_com_juros}')
24
25 # Compreensão
26 lista_com_juros_2 = [valor + (valor * .10) for valor in lista_precos if valor < 300]
27 print(f'Lista triplicada: {lista_com_juros_2}\n')
```

```
-----  
ESTRUTURA DE DADOS: COMPREENSÃO EM LISTAS [ ]  
=====  
Lista triplicada: [110.0, 220.0]  
Lista triplicada: [110.0, 220.0]
```

- a. Crie a lista [1, 2, 3, 5, 6] e depois insira o valor que está faltando na posição correta.
- b. Crie uma lista com 5 números inteiros. Depois imprima a soma desses valores.
- c. Faça um programa que procure na lista numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] e produza:
 - O intervalo de 1 até 9
 - O intervalo de 8 até 13
 - Os números pares
 - Os números ímpares
 - Os múltiplos de 2, 3 e 4
 - Lista reversa
 - O produto dos intervalos 5-6 por 11-12
- d. Faça um programa que preencha uma lista com as notas de 4 alunos, depois imprima a média.
- e. Faça um programa que leia as vogais, depois mostre-as em ordem inversa.
- f. Faça um programa que leia 5 nomes, depois imprima estes nomes com seus respectivos índices.
- g. Ler uma lista com 10 números, depois apresente o maior e o menor número da lista
- h. Faça um programa que leia um número indeterminado de notas (pressione 's' ou '0' para sair).

Após esta entrada de dados, faça o seguinte:

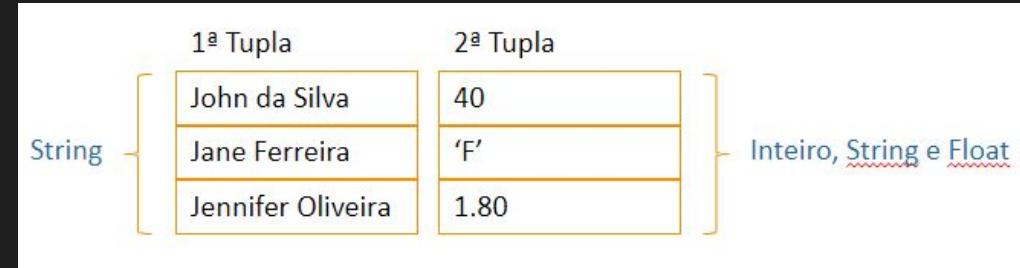
- Mostre a quantidade de notas que foram lidas.
- Exiba todas as notas na ordem em que foram informadas.
- Exiba todas as notas na ordem inversa à que foram informadas, uma abaixo da outra.
- Calcule e mostre a soma das notas.
- Calcule e mostre a média das notas.

- a. Faça um programa que preencha uma lista com 50 números aleatórios. Depois fatie essa lista em 5 listas de 10 elementos.
- b. Utilizando o exercício anterior, coloque todas as listas em ordem crescente de valor
- c. Crie uma lista com 6 nomes, depois verifique se o nome 'Pedro' está nessa lista.
- d. Faça um programa que recebe 7 números inteiros. Depois quebre essa lista em: lista de pares e lista de ímpares.
- e. Faça um programa que gere 10 números aleatórios. Após gerar esses números, crie duas listas novas com ordenação ascendente e descendente.
- f. Faça um programa que sorteia os números da Mega Sena e da Lotofácil

Tupla é uma Lista imutável. O que diferencia a Estrutura de Dados Lista da Estrutura de Dados Tupla é que a primeira pode ter elementos adicionados a qualquer momento, enquanto que a segunda estrutura, após definida, não permite a adição ou remoção de elementos. Devemos pensar num primeiro momento, que a Tupla é uma lista que restringe a adição, alteração, remoção e o ordenamento dos elementos. No entanto, pensar numa tupla como sendo somente uma lista imutável não está totalmente correto.

Características:

- É uma lista declarada como constante
- Não é possível inserir, alterar ou remover elementos (não diretamente)
- Toda tupla será um conjunto de objetos também imutáveis



Quando se utiliza tuplas?

As tuplas são sequências imutáveis, normalmente usadas para armazenar coleções de dados heterogêneos (como a tupla produzida pela função enumerate nativa). As tuplas também são usadas para casos onde uma sequência imutável de dados homogêneos é necessária (como permitir o armazenamento em set ou dict).

As tuplas podem possuir valores de tipos diferentes, onde cada uma representa algo específico. Como o próprio retorno da função enumerate de uma lista, o retorno será uma tupla de dois valores, em que o primeiro representa o índice do valor na lista e o segundo o valor propriamente dito.

ESTRUTURA DE DADOS: TUPLAS ()

```
=====
numeros:      (1, 2, 3, 4, 5, 6)
Atores:       ('Norman Reedus', 'Melissa McBride', 'Lauren Cohan')
personagens: ('Daryl Dixon', 'Carol Peletier', 'Maggie Rhee')
decimais:     (1.2, 3.4, 5.6, 7.8)
Miscelânea:   ('John', 40, 1.77, True)
=====
```

```

5   import os
6
7   # Limpando o terminal
8   os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: TUPLAS ()')
12 print('=' * 70)
13
14 # Declaração
15 numeros = (1, 2, 3, 4, 5, 6) # inteiros
16 atores = ('Norman Reedus', 'Melissa McBride', 'Lauren Cohan') # Strings
17 personagens = ('Daryl Dixon', 'Carol Peletier', 'Maggie Rhee') # Strings
18 decimais = (1.2, 3.4, 5.6, 7.8) # ponto flutuante
19 mix = ('John', 40, 1.77, True)
20
21 # Saída
22 print(f'\tnumeros: \t{numeros}') # \t = Tabulação
23 print(f'\tAtores: \t{atores}')
24 print(f'\tpersonagens: \t{personagens}')
25 print(f'\tdecimais: \t{decimais}')
26 print(f'\tMiscelânea: \t{mix}')
27 print('-'*80)
28 print()
```

O operador `in` verifica se o operando à sua esquerda, está contido na lista a sua direita, da mesma forma que o operador `not in` que verifica o contrário.

Essa é uma maneira para simplificar a verificação se o elemento X está contido na lista Y.

ESTRUTURA DE DADOS: TUPLAS ()

Verificação 1: Estão contidos em números e personagens.

Verificação 2: Não estão contidos em atores e decimais

```

10 print('-' * 80)
11 print('ESTRUTURA DE DADOS: TUPLAS ()')
12 print('=' * 80)
13
14 # Declaração
15 numeros = (1, 2, 3, 4, 5, 6) # inteiros
16 atores = ('Norman Reedus', 'Melissa McBride', 'Lauren Cohan') # Strings
17 personagens = ('Daryl Dixon', 'Carol Peletier', 'Maggie Rhee') # Strings
18 decimais = (1.2, 3.4, 5.6, 7.8) # ponto flutuante
19 mix = ('John', 40, 1.77, True)
20
21 # Condicional
22 # pode testar com o not in
23 if (3 in numeros and 'Maggie Rhee' in personagens):
24     resposta = 'Estão contidos em números e personagens.'
25 else:
26     resposta = 'Não estão cotidos em números e personagens.'
27
28 if ('Maria' not in atores and 1.10 not in decimais):
29     resposta2 = 'Não estão contidos em atores e decimais'
30 else:
31     resposta2 = 'Estão contidos em atores e decimais'
32
33 # Saída
34 print(f'Verificação 1: {resposta}\n') # \n Salta linha
35 print(f'Verificação 2: {resposta2}\n')
36 print('*'*80)
37 print()

```

Varrendo tuplas utilizando estruturas de repetição for in.

ESTRUTURA DE DADOS: TUPLAS ()

Números: 1 2 3 4 5 6

Atores: Norman Reedus Melissa McBride Lauren Cohan

Personagens: Daryl Dixon Carol Peletier Maggie Rhee

Decimais: 1.2 3.4 5.6 7.8

Miscelânea: 1.2 3.4 5.6 7.8

```
10  print('-' * 80)
11  print('ESTRUTURA DE DADOS: TUPLAS ()')
12  print('=' * 80)
13
14  # Declaração
15  numeros = (1, 2, 3, 4, 5, 6)
16  atores = ('Norman Reedus', 'Melissa McBride', 'Lauren Cohan')
17  personagens = ('Daryl Dixon', 'Carol Peletier', 'Maggie Rhee' )
18  decimais = (1.2, 3.4, 5.6, 7.8)
19  mix = ('John', 40, 1.77, True)
20
21  # Saída
22  print('Números: ', end= ' ')
23  for numero in numeros:
24      print(numero, end=' ')
25  print()
26
27  print('Atores: ', end= ' ')
28  for atorAtriz in atores:
29      print(atorAtriz, end=' ')
30  print()
31
32  print('Personagens: ', end= ' ')
33  for personagem in personagens:
34      print(personagem, end= ' ')
35  print()
36
37  # Varrer as tuplas restantes!
38  print('*'*80)
39  print()
```

Varrendo tuplas utilizando estruturas de repetição `for in enumerate()`. A saída será dada pela chave e valor. As chaves são opcionais, use somente se houver a necessidade de indicar a posição de cada valor na tupla.

ESTRUTURA DE DADOS: TUPLAS ()

```
Números:      0 : 1    1 : 2    2 : 3    3 : 4    4 : 5    5 : 6
Atores:       0 : Norman Reedus   1 : Melissa McBride   2 : Lauren Cohan
Personagens:  0 : Daryl Dixon   1 : Carol Peletier   2 : Maggie Rhee
Decimais:     0 : 1.2    1 : 3.4    2 : 5.6    3 : 7.8
Miscelânea:   0 : John    1 : 40    2 : 1.77   3 : True
```

```
10  print('-' * 80)
11  print('ESTRUTURA DE DADOS: TUPLAS ()')
12  print('=' * 80)
13
14  # Declaração
15  numeros = (1, 2, 3, 4, 5, 6)
16  atores = ('Norman Reedus', 'Melissa McBride', 'Lauren Cohan')
17  personagens = ('Daryl Dixon', 'Carol Peletier', 'Maggie Rhee')
18  decimais = (1.2, 3.4, 5.6, 7.8)
19  mix = ('John', 40, 1.77, True)
20
21  # Saída
22  print('Números: \t', end=' ')
23  for indice, numero in enumerate(numeros):
24      print(f'{indice} : {numero}', end='   ')
25  print()
26
27  print('Atores: \t', end=' ')
28  for indice, ator in enumerate(atores):
29      print(f'{indice} : {ator}', end='   ')
30  print()
31
32  # Varrer as tuplas restante!
33
34  print('*'*80)
35  print()
```

Adicionando e removendo elementos de uma tupla.

Os métodos `append()` e `remove()` não funcionam para as tuplas. Neste caso devemos realizar uma conversão para lista.

```

10 print('-' * 80)
11 print('ESTRUTURA DE DADOS: TUPLAS ()')
12 print('=' * 80)
13
14 # Declaração
15 nomes = ()
16
17 # Convertendo a tupla em lista
18 listaNomes = list(nomes)
19
20 # Entrada de Dados
21 for c in range(0, 4, 1):
22     listaNomes.append(str(input('Digite o nome: ')))
23
24 # Convertendo a lista em tupla
25 nomes = tuple(listaNomes)
26
27 # Saída
28 print('Nomes: \t', end=' ')
29 for indice, nome in enumerate(nomes):
30     print(f'{indice} : {nome}', end='      ')
31 print()
32
33 print('*'*80)
34 print()
35

```

```

36 nome = str(input('Nome para remover da Tupla: '))
37 if (nome not in nomes):
38     print('Nome não está na tupla!')
39 else:
40     # Convertendo a tupla em lista
41     listaNomes = list(nomes)
42     listaNomes.remove(nome)
43
44     # Convertendo a lista em tupla
45     nomes = tuple(listaNomes)
46
47 # Saída
48 print('Nomes: \t', end=' ')
49 for indice, nome in enumerate(nomes):
50     print(f'{indice} : {nome}', end='      ')
51 print()
52 print('*'*80)
53 print()

```

ESTRUTURA DE DADOS: TUPLAS ()

```

=====
Digite o nome: Maria
Digite o nome: Pedro
Digite o nome: José
Digite o nome: Ana
Nomes: 0 : Maria    1 : Pedro    2 : José    3 : Ana
=====
```

```

Nome para remover da Tupla: Maria
Nomes: 0 : Pedro    1 : José    2 : Ana
=====
```

Métodos para as tuplas:

`count()`: Retorna o número de vezes que um item aparece

`index()`: Procura por um valor específico e retorna a posição

ESTRUTURA DE DADOS: TUPLAS ()

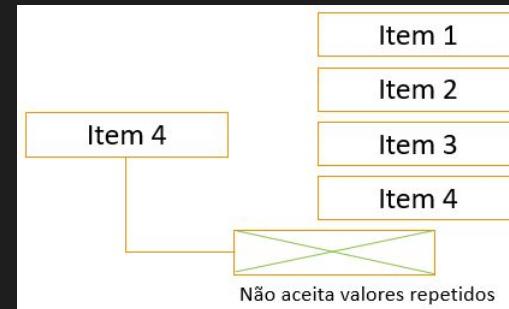
```
Tupla 1: ('Azul', 'Amarelo', 'Verde')
Tupla 3: ('Azul', 10, 1.8, 10, 'Azul', 10)
0 Verde está na posição: 2
0 número 10 aparece 3 vezes
```

```
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: TUPLAS ()')
12 print('=' * 70)
13
14 # Declaração
15 nomes = ()
16
17 minha_tupla_1 = ('Azul', 'Amarelo', 'Verde')
18 minha_tupla_2 = (1, 2, 3)
19 minha_tupla_3 = ('Azul', 10, 1.8, 10, 'Azul', 10)
20
21 posicao = minha_tupla_1.index('Verde')
22 quantidade = minha_tupla_3.count(10)
23
24 print('-' * 70)
25 print(f'Tupla 1: {minha_tupla_1}')
26 print(f'Tupla 3: {minha_tupla_3}')
27 print(f'O Verde está na posição: {posicao}')
28 print(f'O número 10 aparece {quantidade} vezes')
29 print('=' * 70)
30 print()
```

São estruturas disponíveis como builtins do Python, utilizadas para representar coleções desordenadas de elementos únicos. É importante sempre lembrar da teoria de conjuntos. Assim, os itens definidos podem aparecer em uma ordem diferente sempre que você os usa e não podem ser referenciados por índice ou chave. Os exemplos mostrarão como adicionar, atualizar e remover um Set.

Características:

- Estrutura semelhante a lista
- Tem como princípio conter uma lista de valores diferentes
- É uma lista sem itens repetidos
- A set também é imutável



```

5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: SET ') # SET => {}
12 print('=' * 70)
13
14 # Declaração
15 nomes = {'John', 'Jane', 'Carol'}
16 numeros = {1, 2, 3, 4, 5, 6, 7}
17
18 # print(nomes[0]) #Erro, set não tem índice
19
20 # Mas posso varrer com for
21 for nome in nomes:
22     print(nome, end=' ')
23
24 print()
25 print()
26 print('*'*80)
27 print('Adicionando elemento nos Sets')
28 print('*'*80)
29 # numeros.append() # Erro, não tem função append para Sets
30

```

```
31 # método Add() Inserindo itens
32 print()
33 linguagens = {"Python", "Java", "PHP", "C"}
34
35 # Inserindo um elemento no set.
36 print(f'Antes {linguagens}')
37 linguagens.add("C#")
38 print(f'Antes {linguagens}')
39
40 print()
41 print()
42 # Método update()
43 linguagens2 = {"Python", "Java", "PHP", "C"}
44
45 # Inserindo mais de um elemento no set.
46 print(f'Antes {linguagens2}')
47 linguagens2.update(["Smalltalk", "Javascript"])
48 print(f'Depois {linguagens2}')
49 print()
```

ESTRUTURA DE DADOS: TUPLAS ()

John Jane Carol

Adicionando elemento nos Sets

Antes {'PHP', 'Java', 'Python', 'C'}
Antes {'PHP', 'C#', 'C', 'Java', 'Python'}

Antes {'PHP', 'Java', 'Python', 'C'}
Depois {'Javascript', 'PHP', 'Smalltalk', 'C', 'Java', 'Python'}

O método `remove()` remove o elemento especificado do conjunto.

O método `discard()` remove o item especificado do set.

`remove()`

Este método é diferente do `discard()`, porque o `remove()` vai gerar um erro se o item especificado não existir.

`discard()`

Este método é diferente do `remove()`, porque o `discard()` não vai gerar um erro se o item especificado não existir.

ESTRUTURA DE DADOS: SET

Frutas antes:{'Uva', 'Abacaxi', 'Melão', 'Maçã'}
 Frutas depois:{'Uva', 'Abacaxi', 'Melão'}

Frutas antes:{50, 20, 40, 60, 30}
 Frutas depois:{50, 20, 40, 60}

```

10  print('-' * 70)
11  print('ESTRUTURA DE DADOS: SET ') # SET => {}
12  print('=' * 70)
13
14  # método remove()
15  print()
16  frutas = {"Maçã", "Uva", "Abacaxi", "Melão"}
17
18  print('*'*70)
19  print(f'Frutas antes:{frutas}')
20  # coloque um valor diferente para testar o erro
21  frutas.remove('Maçã')
22  print(f'Frutas depois:{frutas}')
23
24  # Método discard()
25  print()
26  numeros = {20, 30, 40, 50, 60}
27
28  print()
29  print('*'*70)
30  print(f'Frutas antes:{numeros}')
31  # coloque um valor diferente para testar o erro
32  numeros.discard(30)
33  print(f'Frutas depois:{numeros}')
34  print()

```

Faça um estudo sobre os métodos utilizados com **Tuplas** descrevendo suas funcionalidades e também demonstrado um exemplo de cada. Em seguida crie um programa — diferente do representado no documento Word — para exemplificar as funcionalidades.

Faça um estudo sobre os métodos utilizados com **Sets** descrevendo suas funcionalidades e também demonstrado um exemplo de cada. Em seguida crie um programa qualquer (diferente do representado no documento Word) para exemplificar um mínimo de 3 e máximo de 5 funcionalidades ao mesmo tempo.

Observações:

- Tentem implementar condicionais, estruturas de dados e entradas do usuário
- Validações são opcionais
- No documento, formate o texto com fonte Arial 12 normal para o texto e courier new para os exemplos dos códigos.
- O documento deverá conter um cabeçalho com nome do aluno, turma e data
- Data de entrega 20/01/2023 - 20h

Envio:

Um documento do MS Word com os dois estudos

Um programa python englobando no mínimo 2 métodos para as Tuplas e 3 métodos para os Sets.

Um dicionário se parece com uma lista, mas é mais geral. Em uma lista, os índices têm que ser números inteiros; em um dicionário, eles podem ser de (quase) qualquer tipo.

Um dicionário contém uma coleção de índices, que se chamam **chaves** e uma coleção de **valores**. Cada chave é associada com um único valor. A associação de uma chave e um valor chama-se par chave-valor ou item.

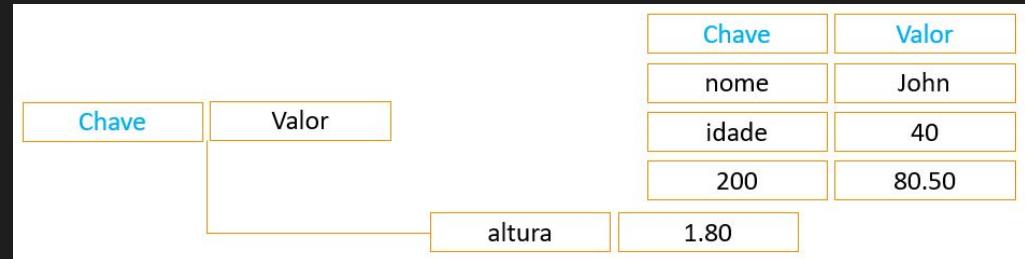
Em linguagem matemática, um dicionário representa um mapeamento de chaves a valores, para que você possa dizer que cada chave “mostra o mapa” a um valor.

A função `dict` cria um novo dicionário sem itens. Como `dict` é o nome de uma função integrada, você deve evitar usá-lo como nome de variável.

```
meu_dicionario = dict()
```

As chaves {} representam um dicionário vazio. Para acrescentar itens ao dicionário, você pode usar colchetes:

```
meu_dicionario = {}  
meu_dicionario['nome'] = 'John Doe'  
meu_dicionario['idade'] = 50  
meu_dicionario['peso'] = 74.5
```



```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
12 print('=' * 70)
13
14 # Declaração
15 compras = {}
16 pessoas = {}
17 cores = {}
18 elementos = dict()
19 numeros = dict()
20
21 # Atribuindo valores
22 compras['id'] = 1
23 compras['item'] = 'Caderno'
24 compras['valor'] = 10.80
25
26 pessoas['id'] = '0010'
27 pessoas['nome'] = 'Sherlock Holmes'
28 pessoas['endereco'] = 'Baker Street'
29 pessoas['numero'] = '221B'
30 pessoas['cidade'] = 'Londres'
31 pessoas['país'] = 'Inglaterra'
32

33 cores['red'] = 'Vermelho'
34 cores['green'] = 'Verde'
35 cores['blue'] = 'Azul'
36
37 elementos['Pb'] = 'Chumbo'
38 elementos['Au'] = 'Ouro'
39 elementos['N'] = 'Nitrogênio'
40
41 numeros[1] = 100
42 numeros[2] = 200
43 numeros[3] = 300
44
45 # Saída simples
46 print(f'Minhas compras: {compras}')
47 print(f'Detectives: {pessoas}')
48 print(f'Core RGB: {cores}')
49 print(f'Tabela periódica: {elementos}')
50 print(f'Lista de números: {numeros}')
51 print()
52 print('*'*100)

```

Declarando dicionários

```

Minhas compras: {'id': 1, 'item': 'Caderno', 'valor': 10.8}
Detectives: {'id': '0010', 'nome': 'Sherlock Holmes', 'endereco': 'Baker Street', 'nu
Core RGB: {'red': 'Vermelho', 'green': 'Verde', 'blue': 'Azul'}
Tabela periódica: {'Pb': 'Chumbo', 'Au': 'Ouro', 'N': 'Nitrogênio'}
Lista de números: {1: 100, 2: 200, 3: 300}

```

Erros em declarações de dicionários:

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
12 print('=' * 70)
13
14 # índices iguais
15 dicionario1 = {'nome': 'John', 'nome': 'Jane'}
16
17 # Posso ter uma tupla como índice e uma lista como valor
18 dicionario2 = {(1, 2) : [1, 2]}
19
20 # Mas não posso ter uma lista como índice e uma tupla como valor
21 dicionario3 = {[1, 2] : (1, 2)}
22
23 # Saída
24 print('*'*80)
25 print(dicionario1)
26 print(dicionario2)
27 print(dicionario3)
28 print()
```

```
Traceback (most recent call last):
File "d:\Trabalho\Senac\Turma 0277\UC10\python\dicionario3.py", line 21, in <module>
    dicionario3 = {[1, 2] : (1, 2)}
          ^
TypeError: unhashable type: 'list'
```

`len()`: Este método retorna o número de elementos de um dicionário

Faltou `del()`

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
12 print('=' * 70)
13
14 # declaração
15 agenda = {
16     'Jojo': '99196-3030',
17     'Dio': '99196-5050',
18     'Jolyne': '99196-6060',
19     'Lisa Lisa': '99196-7070',
20     'Speedwagon': '99196-8080',
21     'Zeppeli': '99196-9090',
22     'Suzie Q': '99196-0000'
23 }
24
```

```
25 # Apagando elemento da agenda
26 del(agenda['Speedwagon'])
27 del(agenda['Dio'])
28 del(agenda['Zeppeli'])
29
30 # Verificando o tamnho da agenda
31 tamanho = len(agenda)
32
33 # Saída
34 print('*' * 80)
35 print(agenda)
36 print(f'Tamanho da agenda: {tamanho}')
37 print()
```

ESTRUTURA DE DADOS: DICIONÁRIO

```
=====
{ 'Jojo': '99196-3030', 'Jolyne': '99196-6060', 'Lisa Lisa': '99196-7070'
Tamanho da agenda: 4
```

`del()`: este método apaga um item do dicionário, basta passar a **chave** como parâmetro

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 70)
11 print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
12 print('=' * 70)
13
14 # declaração
15 agenda = {
16     'Jojo': '99196-3030',
17     'Dio': '99196-5050',
18     'Jolyne': '99196-6060',
19     'Lisa Lisa': '99196-7070',
20     'Speedwagon': '99196-8080',
21     'Zeppeli': '99196-9090',
22     'Suzie Q': '99196-0000'
23 }
24 d = 'Suzie Q'
25
26     # Pegando um elemento do Dicionário
27 del(agenda[d])
28 del(agenda['Dio'])
29 del(agenda['Zeppeli'])
30
31     # Verificando o tamanho da agenda
32 tamanho = len(agenda)
33
34     # Saída
35 print()
36 print(agenda)
37 print(f'Tamanho da agenda: {tamanho}')
38 print()
-----  

ESTRUTURA DE DADOS: DICIONÁRIO  

=====  

Antes:  

{'Jojo': '99196-3030', 'Dio': '99196-5050', 'Jolyne': '99196-6060', 'Lisa Lisa': '99196-7070', 'Speedwagon': '99196-8080', 'Zeppeli': '99196-9090', 'Suzie Q': '99196-0000'}  

Depois:  

{'Jojo': '99196-3030', 'Jolyne': '99196-6060', 'Lisa Lisa': '99196-7070', 'Speedwagon': '99196-8080'}
-----
```

`Keys()`: este método retorna as chaves (índices) do dicionário.

`values()`: este método retorna os valores do dicionário.

`items()`: este método retorna as chaves e valores do dicionário

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 # Limpando o terminal
8 os.system('cls')
9
10 print('-' * 120)
11 print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
12 print('=' * 120)
13
14 # declaração
15 agenda = {
16     'nome': 'John Doe',
17     'cpf': '123.456.789-12',
18     'cep': '36036-630',
19     'celular': '(32)9.9196-7070',
20 }
21
22 # pegando chaves, valores e itens
23 chaves = agenda.keys()
24 valores = agenda.values()
25 itens = agenda.items()
26
27 # Verificando o tamanho da agenda (para lembrar)
28 tamanho = len(agenda)
29
30 # Saída
31 print('-'*120)
32 print(f'- Todas as chaves da agenda: \n{chaves}')
33 print()
34 print(f'- Todos os valores da agenda: \n{valores}')
35 print()
36 print(f'- Todos os itens da agenda: \n{itens}')
37 print('-' * 120)
38 print()

```

ESTRUTURA DE DADOS: DICIONÁRIO

- Todas as chaves da agenda:
`dict_keys(['nome', 'cpf', 'cep', 'celular'])`

- Todos os valores da agenda:
`dict_values(['John Doe', '123.456.789-12', '36036-630', '(32)9.9196-7070'])`

- Todos os itens da agenda:
`dict_items([('nome', 'John Doe'), ('cpf', '123.456.789-12'), ('cep', '36036-630'), ('celular', '(32)9.9196-7070')])`

`get()`: esse método retorna um elemento do dicionário, essa é a forma mais utilizada.

```
10  print('-' * 70)
11  print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
12  print('=' * 70)
13
14  # Declaração
15  agenda = {
16      'Rick': 991964087, 'Daryl': 999890000,
17      'Carol': 984564087, 'Glen': 984555050,
18      'Michone': 988885050
19  }
20
21  # Pegando um elemento do Dicionário
22  busca_1 = agenda.get('Michone', 'Contato não encontrado')
23  busca_2 = agenda.get('Daryl', 'Contato não encontrado')
24  busca_3 = agenda.get('Negan', 'Contato não encontrado')
25
26  # Saída
27  print()
28  print(f'O celular da Michone: {busca_1}')
29  print(f'O celular do Daryl: {busca_2}')
30  print(f'O celular do Negan: {busca_3}')
31  print('-' * 70)
32  print()
```

ESTRUTURA DE DADOS: DICIONÁRIO

0 celular da Michone: 988885050
0 celular da Daryl: 999890000
0 celular da Negan: Contato não encontrado

popitem(): esse método remove o último elemento do dicionário. Retorna uma Tupla

```
9  print('-' * 70)
10 print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
11 print('=' * 70)
12
13 agenda = {
14     'Rick': 991964087,
15     'Daryl': 999890000,
16     'Carol': 984564087,
17 }
18
19 # Antes
20 print('- antes:')
21 print(agenda)
22
23 # copia a chave e o valor e exclui do dicionário origem
24 nova_agenda = agenda.popitem()
25 print()
26
27 # Depois
28 print('- depois:')
29 print(agenda)
30 print()
31
32 # Imprimindo a tupla
33 print(f'Tupla retornada: {nova_agenda}')
34 print('=' * 70)
35 print()
```

```
ESTRUTURA DE DADOS: DICIONÁRIO
=====
- antes:
{'Rick': 991964087, 'Daryl': 999890000, 'Carol': 984564087}

- depois:
{'Rick': 991964087, 'Daryl': 999890000}

Tupla retornada: ('Carol', 984564087)
```

in e not in: verifica se o elemento está ou não contido no dicionário. Retorno True ou False.

```
9  print('-' * 70)
10 print('ESTRUTURA DE DADOS: DICIONÁRIO ') # dict => {}
11 print('=' * 70)
12
13 agenda = {
14     'Jojo': '99196-3030',
15     'Dio': '99196-5050',
16     'Jolyne': '99196-6060',
17     'Lisa Lisa': '99196-7070',
18     'Speedwagon': '99196-8080',
19     'Zeppeli': '99196-9090',
20     'Suzie Q': '99196-0000'
21 }
22 print()
23 print(agenda)
24 print()
25
26 if 'Jojo' in agenda:
27     print('Primeiro teste: verificando se Jojo está no Dicionário')
28     print('VERDADEIRO! Jojo está no dicionário')
29 else:
30     print('FALSO! Jojo não está no dicionário')
31
32 print()
33
34 if 'John' not in agenda:
35     print('Segundo teste: verificando se John não está no dicionário')
36     print('VERDADEIRO! John não está no dicionário')
37 else:
38     print('FALSO: John está no dicionário')
39 print('-' * 70)
40 print()
```

ESTRUTURA DE DADOS: DICIONÁRIO
=====

{'Jojo': '99196-3030', 'Dio': '99196-5050', 'Jolyne': '99196-6060', 'Lisa Lisa': '99196-7070', 'Speedwagon': '99196-8080', 'Zeppeli': '99196-9090', 'Suzie Q': '99196-0000'}

Primeiro teste: verificando se Jojo está no Dicionário
VERDADEIRO! Jojo está no dicionário

Segundo teste: verificando se John não está no dicionário
VERDADEIRO! John não está no dicionário

update(): Esta função coloca elementos de um dicionário em outro. As informações do primeiro dicionário são substituídas.

```
9  print('-' * 80)
10 print('ESTRUTURA DE DADOS: DICIONÁRIO ')
11 print('=' * 80)
12
13 agenda_1 = {
14     'nome': 'John Doe',
15     'endereco': 'Rua Marechal, 166',
16     'celular': '99196-3030',
17 }
18
19 print()
20 print('- Antes da atualização')
21 print(agenda_1)
22
23 # Criando uma segunda agenda
24 agenda_2 = {
25     'nome': 'Jane Doe',
26     'endereco': 'Rua Halfeld, 10',
27     'celular': '98817-0000'}
28
29 # Atualizando o primeiro diconário com os dados do segundo
30 agenda_1.update(agenda_2)
31
32 print()
33 print('- Depois da atualização')
34 print(agenda_1)
35 print('=' * 80)
36 print()
```

ESTRUTURA DE DADOS: DICIONÁRIO

- Antes da atualização
{'nome': 'John Doe', 'endereco': 'Rua Marechal, 166', 'celular': '99196-3030'}

- Depois da atualização
{'nome': 'Jane Doe', 'endereco': 'Rua Halfeld, 10', 'celular': '98817-0000'}

Erro

copy(): Retorna uma cópia de um dicionário

```
9  print('-' * 80)
10 print('ESTRUTURA DE DADOS: DICIONÁRIO ')
11 print('=' * 80)
12
13 elementos = {} # Dicionário
14 periodica = [] # lista
15
16 for c in range(0, 2):
17     # entrada de dados com for
18     elementos['simbolo'] = str(input('Símbolo do elemento: '))
19     elementos['nome'] = str(input('Nome do elemento: '))
20
21     # Usa o append() com o copy() para fazer uma cópia do dicionário
22     periodica.append(elementos.copy())
23     elementos.clear() # limpa a lista
24
25 print()
26 print('*'*80)
27 print(periodica)
28 print('*'*80)
29
30 # For aninhado para percorrer a lista e o dicionário
31 for elemento in periodica: # para cada elemento na minha lista
32     for chave, valor in elemento.items(): # para cada chave e valor dos meus elementos
33         print(f'{chave} : {valor}', end=' ' + '\n') # Imprima a chave e o valor
34 print()
35 print('*'*80)
36 print()
```

ESTRUTURA DE DADOS: DICIONÁRIO

```
=====
Símbolo do elemento: Na
Nome do elemento: Sódio
Símbolo do elemento: Au
Nome do elemento: Ouro
```

```
[{'simbolo': 'Na', 'nome': 'Sódio'}, {'simbolo': 'Au', 'nome': 'Ouro'}]
```

```
simbolo : Na
nome : Sódio
simbolo : Au
nome : Ouro
```

- a. Faça um programa para criar um dicionário com 4 elementos.
- b. Utilizando o exercício anterior, adicione mais 2 elementos ao dicionário.
- c. Utilizando o exercício anterior , retire um elemento do dicionário.
- d. Faça um programa para criar um dicionário com 5 cores. Depois, imprima as chaves e os valores deste dicionário.
- e. Faça um programa para criar um dicionário com 5 ferramentas. Depois, imprima os valores e a quantidade de elementos do dicionário.
- f. Faça um programa que cadastre 5 tipos de vinho. Para os campos deste cadastro tome como referência nome do vinho, tipo, teor alcoólico, e safra.
- g. Faça um programa para cadastrar os alunos de uma escola. Para os campos, tome como referência o nome do aluno, data de nascimento e matrícula.
- h. Faça um programa para ler o dicionário nomes = {'nome': 'John', 'idade': 40, 'peso': 80, 'altura': 1.70}. Em seguida realize as seguintes ações:
 - Listar chaves e valores com laço - Deletar o peso
 - Listar novamente chaves e valores - mostrar o nome e altura
- i. Faça um programa para criar um dicionário com 4 elementos. Depois imprima a lista completa, delete o último elemento e exiba uma listagem nova.
- j. Faça um programa para criar um dicionário com nomes de frutas. Em seguida verifique se tem abacaxi nos valores.
- k. Faça um programa que peça continuamente o nome e a idade de uma pessoa. Caso o usuário digite a idade 999, o programa será finalizado e executará uma impressão com os nomes cadastrados.

Entrega: 29/01/2023 - 23:59

Funções são blocos de código independentes que realizam determinadas tarefas que normalmente precisam ser executadas diversas vezes dentro de uma aplicação.

Quando surge essa necessidade, para que várias instruções não precisem ser repetidas, elas são agrupadas em uma função, à qual é dado um nome e que poderá ser chamada/executada em diferentes partes do programa.

Partes de uma função em python: nome, parâmetros, argumento e corpo.

Para declarar uma função é preciso usar o comando **def** e em seguida o nome da função em minúsculo. Em nomes compostos, usar o underline.

ESTUDO DE FUNÇÕES

Estou estudando funções em Python 3+

A função olá mundo foi executada!

```

1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: DECLARAÇÃO')
11 print('=' * 70)
12
13 # Para declarar uma função é preciso usar o comando def
14 # em seguida o nome da função em minúsculo
15 # nomes compostos, usar o underline, finalizando com dois pontos(:)
16
17
18 def ola_mundo():
19
20     # Aqui começa o bloco da função
21
22     print('Estou estudando funções em Python 3+')
23     print('-'*70)
24
25
26 # Programa principal
27 # chamando a função
28 ola_mundo()
29 print('A função olá mundo foi executada!')
30 print()
```

É muito comum passarmos informações para as funções. Essas informações são processadas dentro da função e, podendo ou não, retornar um valor.

Os parâmetros são declarados dentro dos parênteses e quando houver mais de um, nós utilizamos vírgula para separá-los.

Parâmetros ou argumentos?

Parâmetros: quando definimos uma função

Argumento: quando invocamos uma função

FUNÇÃO: PARÂMETROS E ARGUMENTOS

```
A soma de 2 + 3 = 5
A Subtração de 10 - 5 = 5
```

```

5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: PARÂMETROS E ARGUMENTOS')
11 print('=' * 70)
12
13
14 def soma(a, b):
15
16     soma = a + b
17     print(f'A soma de {a} + {b} = {soma}')
18
19
20 def subtracao(a, b):
21     subtracao = a - b
22     print(f'A Subtração de {a} - {b} = {subtracao}')
23
24
25 # duas linhas depois: invocando a função
26 # e passando 2 argumentos
27 x = 2
28 y = 3
29 soma(x, y)
30 subtracao(10, 5)
31 print('*' * 70)
32 print()

```

Na declaração da função dentro dos parênteses, podemos passar valores junto com as variáveis de parâmetro. Esse tipo de passagem é chamado de **parâmetro default/opcional**. Esses parâmetros não são obrigatórios, mas se utilizarmos, devemos deixá-los para o final da declaração.

Nada nos impede de passarmos argumentos na hora de invocarmos a função. Mas, relembrando, os parâmetros sem argumentos devem preceder os que utilizam argumentos.

ESTUDO DE FUNÇÕES

```
Bem vindo ao sistema acadêmico, aluno
```

```
Bem vindo ao sistema acadêmico, John
```

```
5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: PARÂMETROS OPCIONAIS')
11 print('=' * 70)
12
13 # Função
14 def login_aluno(login = 'aluno', senha = 123456):
15     print('*' * 70)
16     print(f'Bem vindo ao sistema acadêmico, {login}')
17     print('*' * 70)
18
19
20 # chamando a função
21 login_aluno()
22
23 # chamando a função com argumentos
24 login_aluno('John', 343434)
```

Parâmetros Posicionais

Os argumentos são recebidos na ordem que são passados pela função. É a forma mais comum de passagem e parâmetros em funções. Também podemos ver que quando usamos os parâmetros posicionais estamos enviando para a função uma Tupla.

```
def funcao(param1, param2, param3):
```

Parâmetros Nomeados

Declaramos uma função colocando valores nos parâmetros. Mas os argumentos, na chamada da função, não precisam seguir a ordem declarada nos parâmetros da função. Neste caso, vemos que os argumentos são enviados como um Dicionário.

```
def funcao(param1='Valor', param2=valor, param3=valor):
```

```
-----  
FUNÇÃO: PARÂMETROS NOMEADOS  
=====
```

```
Bem vindo ao sistema médico, Jane!  
Ano de nascimento: 1980  
Peso: 75  
Altura: 1.9
```

```
5 import os  
6  
7 os.system('cls')  
8  
9 print('-' * 70)  
10 print('FUNÇÃO: PARÂMETROS NOMEADOS')  
11 print('=' * 70)  
12  
13  
14 def dados_paciente(nome='John', nascimento=1970, peso=80, altura=1.80):  
15     print('*'*70)  
16     print(f'Bem vindo ao sistema médico, {nome}!')  
17     print(f'Ano de nascimento: {nascimento}')  
18     print(f'Peso: {peso}')  
19     print(f'Altura: {altura}')  
20     print('*'*70)  
21  
22  
23 # Utilizando parâmetros nomeados não há a necessidade  
24 # de obedecer a ordem na entrada dos argumentos  
25  
26 dados_paciente(nome='Jane', altura=1.90, peso=75, nascimento=1980)  
27  
28 # Mas não podemos mesclar posicional com nomeado  
29 # a linha abaixo vai gerar um erro  
30  
31 dados_paciente(nome='Jane', altura=1.90, peso=75, 1980)
```

Funções com retorno

Essas funções ao invés de apenas mostrar o resultado, elas podem retornar valores que foram obtidos, por exemplo, no processo de algum cálculo. Para retornarmos os valores utilizamos a instrução `return`.

FUNÇÃO: RETORNO DE VALOR

```
Entre com a distância percorrida: 200
Entre com o tempo gasto: 10
```

```
Distância percorrida: 200.0m
Tempo gasto: 600.0min
A velocidade média do veículo é de 20.00km/h
```

```

5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: RETORNO DE VALORES')
11 print('=' * 70)
12
13
14 def velocidade_media(espaco, tempo):
15     # Vm = ΔS/Δt
16     vm = espaco / tempo
17
18     return vm # retorna o cálculo acima
19
20
21 print('-' * 70)
22 print('FUNÇÃO: RETORNO DE VALOR')
23 print('=' * 70)
24
25
26 espaco = float(input('Entre com a distância percorrida: '))
27 tempo = float(input('Entre com o tempo gasto: '))
28
29 calculo_vm = velocidade_media(espaco, tempo)
30
31 print('*70)
32 print(f'Distância percorrida: {espaco}m')
33 print(f'Tempo gasto: {tempo*60}min')
34 print(f'A velocidade média do veículo é de {calculo_vm:.2f}km/h')
35 print('*70)
36 print()
```

Funções com retorno múltiplo de valor

Como o próprio nome diz, essas funções retornam mais de um valor ao mesmo tempo. Geralmente são retornadas listas e tuplas. Mas não existe uma notação exata.

Packing and Unpacking Arguments in Python

Usamos dois operadores `*` (para tuplas) e `**` (para dicionários).

Considere uma situação em que temos uma função que recebe quatro argumentos. Queremos fazer uma chamada para esta função e temos uma lista de tamanho 4. Se simplesmente passarmos uma lista para a função, a chamada não funcionará.

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: PACKING E UNPACKING')
11 print('=' * 70)
12
13
14 def teste(a, b, c, d):
15     print(a, b, c, d)
16
17
18 minha_lista = [1, 2, 3, 4]
19
20 # não funciona
21 teste(minha_lista)
```

Desempacotamento

Podemos usar * no desempacotamento da lista para que todos os elementos dela possam ser passados como parâmetros diferentes.

Observação importante:

Os argumentos e parâmetros devem ter a mesma quantidade de elementos.

```
1 # Curso Técnico de Informática
2 # Autor: Sebastião Marcos
3 # Data: 06/12/2022
4
5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: PACKING E UNPACKING')
11 print('=' * 70)
12
13
14 def teste(a, b, c, d):
15     print(a, b, c, d)
16
17
18 minha_lista = [1, 2, 3, 4]
19
20
21 teste(*minha_lista) # desempacotando a lista
```

Empacotamento

Quando não sabemos quantos argumentos precisam ser passados para uma função python, podemos usar o Packing para empacotar todos os argumentos em uma tupla.

A função `minha_soma()` faz 'packing' para empacotar todos os argumentos que essa chamada de método recebe em uma única variável.

Uma vez que tenhamos essa variável "embalada", podemos fazer coisas com ela que faríamos com uma tupla normal. `args[0]` e `args[1]` lhe dariam o primeiro e o segundo argumentos, respectivamente. Como nossas tuplas são imutáveis, você pode converter a tupla `args` em uma lista para que você também possa modificar, excluir e reorganizar os itens.

```
5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: PACKING E UNPACKING')
11 print('=' * 70)
12
13
14 def minha_soma(*args): # função para somar números
15     return sum(args)
16
17
18 print(minha_soma(1, 2, 3, 4, 5))
19 print(minha_soma(10, 20))
```

Para lembrar:

Packing o * é no parâmetro

Unpacking o * é no argumento

FUNÇÃO: PACKING E UNPACKING

Segunda-feira Terça-feira Quarta-feira

```
5 import os
6
7 os.system('cls')
8
9 print('-' * 70)
10 print('FUNÇÃO: PACKING E UNPACKING')
11 print('=' * 70)
12
13
14 def teste(a, b, c):
15     print(a, b, c)
16
17
18 def fun2(*args): # Empacotando
19
20     args = list(args) # converte a tupla em lista
21
22     args[0] = 'Segunda-feira' # alterando valor
23     args[1] = 'Terça-feira' # alterando valor
24
25     teste(*args) # Desempacotando
26
27
28 # Driver code
29 fun2('trocar', 'trocar', 'Quarta-feira')
30 print('=' * 70)
```

Nas aulas anteriores vimos o uso dos `args(*)` para trabalharmos com listas e tuplas. Agora veremos o tratamento com `keywargs(**)` para a implementação de dicionários na passagem de valores nas funções

```

5   import os
6   from funcoes_uteis import limpa_tela as lt
7   from funcoes_uteis import linha_simples as ls
8   from funcoes_uteis import linha_dupla as ld
9
10
11  def tupla(*tupla):
12      print(tupla)
13
14
15  def lista(*lista):
16      print(lista)
17
18
19  def dicionario(**dicionario):
20      print(dicionario)
21
22
23  os.system('cls')
24
25  print('-' * 70)
26  print('FUNÇÃO: PACKING E UNPACKING')
27  print('=' * 70)
28
29  # limpar a tela
30  lt()
31

32  # inserir uma linha dupla
33  ld(70)
34  print('Funções variádicas')
35  ls(70)
36
37  # imprimindo uma tupla qualquer
38  print('-- Tupla retorna tupla!')
39  tupla('um', 'dois', 'três', 'quatro')
40  print()
41
42  # imprimindo uma lista qualquer
43  print('-- Lista retorna tupla!')
44  lista(1, 2, 3, 4)
45  print()
46
47  # imprimindo um dicionário qualquer
48  print('-- Dicionário retorna dicionário!')
49  dicionario(domingo=1, segunda=2, terca=3, quarta=4)
50
51  # inserir uma linha simples
52  ls(70)
53  print()

=====
Funções variádicas
-----
-- Tupla retorna tupla!
('um', 'dois', 'três', 'quatro')

-- Lista retorna tupla!
(1, 2, 3, 4)

-- Dicionário retorna dicionário!
{'domingo': 1, 'segunda': 2, 'terca': 3, 'quarta': 4}
-----
```

O que são?

Módulos são arquivos de código Python cuja interface é conhecida e que podem ser importados por outros módulos (daremos uma definição formal mais tarde). Dizer que a "interface é conhecida" significa que quando um programador importar um módulo, ele saberá (ou tem meios de saber) quais funções e classes o módulo possui. Ele também saberá como usá-las, isto é, conhecendo seus nomes, parâmetros, que exceções pretende tratar, dentre outras características.

A saber:

Pacote: um conjunto de módulos (pasta)

Módulo: Um arquivo Python isolado

Um conjunto de módulos pode ser guardado dentro de um pacote.

Modularização é o ato de transformar um programa grande em pequenas partes.

Benefícios: Reuso, Legibilidade e manutenção do sistema

O Python possui um conjunto de módulos chamado de Biblioteca Padrão do Python, e nós já utilizamos algumas delas.

Características dos Módulos

Reutilização de código

Podemos usar uma estrutura declarada, desde que a mesma esteja no escopo global do módulo

Namespace ([python avançado, pesquisar](#))

Um namespace é basicamente um sistema para certificar-se que todos os nomes em um programa são únicos e podem ser usados sem qualquer conflito. Você já deve saber que tudo em Python — como sequências de caracteres, listas, funções, etc. — é um objeto. Outro fato interessante é que o Python implementa namespaces como dicionários. Há um mapeamento de nome-para-objeto, com os nomes como chaves e os objetos como valores. Vários namespaces podem usar o mesmo nome e mapeá-los para um objeto diferente.

Namespace local

Este namespace inclui nomes locais dentro de uma função. Ele é criado quando uma função é chamada, e só dura até que a função retorne.

Namespace global

Este namespace inclui nomes de vários módulos importados que você está usando em um projeto. Ele é criado quando o módulo está incluído no projeto, e dura até o script terminar.

Namespace interno

Este namespace inclui funções internas e nomes de exceções internas.

Compartilhamento de estruturas de dados

Podemos criar um componente e compartilhar com os diversos módulos de nossa aplicação

Observação: Nos módulos é comum nos referirmos às estruturas de classes, funções, variáveis etc, chamando-as de símbolos ou nome.

Características dos Módulos

Reutilização de código

Importação de módulo está relacionado a reutilização de código. Para ter acesso a qualquer módulo que não estamos implementando, precisamos importá-los. Para esse processo utilizamos a instrução import.

Depois de importado, podemos acessar os atributos do módulo usando a notação abaixo:

modulo.attr

Objeto
Propriedade



```
1 # Usamos o comando mais o nome da função
2 # para adicionar o módulo na tabela de símbolos local
3
4 import math
5
6 # Podemos acessar um atributo da função math pelo nome do módulo
7 # Assim, enviamos copiamos seu nome para a tabela de símbolos que
8 # estamos trabalhando
9
10 # O número "e" é uma constante matemática que é a base dos logaritmos naturais.
11 e = math.e
12 pi = math.pi
13
14 # Saída
15 print(f'Imprimindo o valor de "e": {e}')
16 print(f'Imprimindo o valor de "pi": {pi}')
```

Características dos Módulos

Às vezes não queremos importar todos os símbolos dentro de um módulo, ou seja, importamos daquele módulo somente o que estamos precisando. Não é aconselhável essa prática, por em algum momento poderá ocorrer algum conflito com os nomes do módulos.

```
from math import sqrt
```

Podemos também importar o módulo e dar a ele um apelido (alias) utilizando a instrução “as”.

```
import modulo as M
from modulo import simbolo as S
from modulo import *
```

Os símbolos podem ser importados para dentro de uma função. Quando a função terminar, ele será removido da memória, como acontece com os elementos locais.

```
1  import os
2
3  # Importando os símbolos
4  from math import pi, e
5
6  # Importando e utilizando um apelido para o módulo
7  from math import sqrt as rq
8
9  # importando todos os símbolos da biblioteca
10 from math import *
11
12 # limpando o terminal
13 os.system('cls')
14
15 # Saída
16 print(f'{e:.2f} | {pi:.2f} | {rq(9)})')
17
18 # importando um símbolo dentro de uma função
19 def fatorial():
20     from math import factorial
21     print(f'Fatorial de 4: {factorial(4)})')
22
23
24 # Saída Fatorial
25 print()
26 fatorial()
27 print()
```

2.72 | 3.14 | 3.0

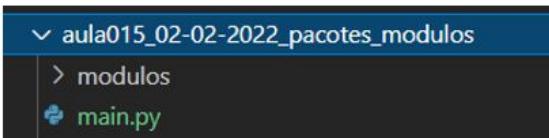
Fatorial de 4: 24

Praticando...

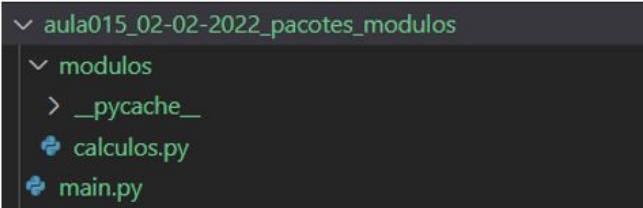
Em sua pasta de projetos Python, crie uma pasta chamada '**modulos**'.

Dentro da pasta de projetos Python, crie um arquivo **python** chamado '**main.py**'

Sua tela deverá estar como na figura abaixo:



Dentro da pasta módulos, crie um arquivo chamado **calculos.py** com as seguintes linhas de código:



```
1  def soma(a, b):
2      s = a + b
3      return s
```

```
1  import os
2
3  # importando o módulo cálculos
4  from modulos.calculos import soma
5
6
7  # Código principal
8  a = int(input('Digite o valor de "a": '))
9  b = int(input('Digite o valor de "b": '))
10
11 # Chamando a função
12 resultado = soma(a, b)
13
14 # Saída
15 os.system('cls')
16 print(f'A soma de {a} + {b} = {resultado}')
17 print()
```

No arquivo **main.py** digite o código ao lado:

módulo

```
main.py  
soma()  
subt()  
multiplica()  
divide()  
...
```

import

módulo

```
calculos.py  
def soma()  
def subt()  
def multiplica()  
def divide()  
...
```

Quando nossos módulos ficam muito grandes, precisamos separá-los para melhor legibilidade e reutilização de código. E quando esses módulos separados ficam muito grandes, criamos pacotes, geralmente nomeados pela sua utilidade.

módulo

```
main.py  
soma()  
subt()  
multiplica()  
divide()  
...
```

import

pacote calculos

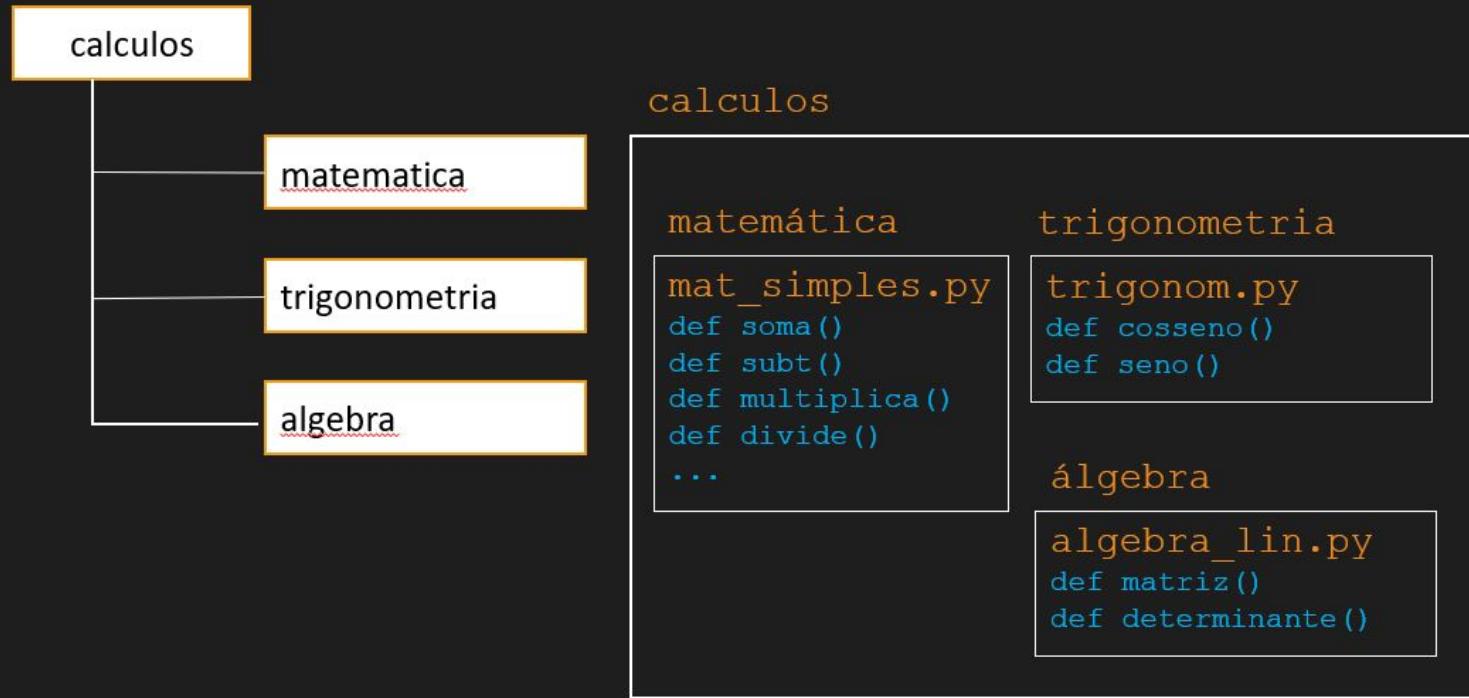
```
mat_simples.py  
def soma()  
def subt()  
def multiplica()  
def divide()  
...
```

```
trigonometr.py  
def cosseno()  
def seno()
```

```
algebra_lin.py  
def matriz()  
def determinante()
```

Pasta

Organizando melhor os códigos



- a. Crie uma função que receba uma lista de números. Depois retorne duas listas com os números pares, os números ímpares, a quantidade de números pares e a quantidade de números ímpares.
- b. Crie uma função que cadastre o nome de uma aluno, a matrícula e a data de nascimento. Depois imprima os resultados cadastrados utilizando uma estrutura de repetição for.
- c. Crie uma função que verifica se uma aluno está cadastrado ou não em um dicionário. Se estiver cadastrado, imprima o nome desse aluno e o resto dos seus dados. O dicionário deverá conter nome, matrícula e a data de nascimento do aluno.
- d. Crie uma função que receba uma temperatura em fahrenheit e retorne o valor em graus célsius.
- e. Crie uma função que receba a altura e o peso de uma pessoa. Depois retorne o seu IMC.
- f. Crie uma função que receba 2 listas:
 - lista 1: nome, peso, idade
 - lista 2: John, 40, 1.8
 - Sua função deverá criar um dicionário contendo chave e valor utilizando como base essas duas listas. Depois, seu programa deverá imprimir esse dicionário utilizando uma estrutura de repetição for.
- g. Crie um programa que peça ao usuário 2 números maiores que 0 e menores que 11. Depois mostre um menu com as seguintes operações:
 1. Soma
 2. Subtração
 3. Produto
 4. Divisão
 4. Divisão inteira
 6. Resto da divisãoO usuário deverá escolher um número maior ou igual a 1 e menor ou igual a 6. Em seguida, você criará funções que retornem os resultados das operações, imprimindo os resultados na tela.
- h. Uma Academia deseja fazer uma pesquisa entre seus clientes para descobrir a média de altura e peso de seus clientes. Faça um programa que pergunte a cada um dos clientes da academia seu código, nome, altura e peso. Após esse cadastramento, seu programa deverá listar os dados dos clientes e a média. Para sair do programa o usuário deverá digitar o valor zero(0). O número de clientes é indefinido. Veja a saída no próximo slide.
- i. Faça um programa de perguntas e respostas sobre os estados e capitais brasileiras. O programa deverá exibir para o usuário o está e perguntar qual a sua capital. Se o usuário errar a resposta, o programa será finalizado apresentando quantas perguntas estão corretas. Utilize ao menos uma função e não há a necessidade de modularizar o código.

PROGRAMA: ACADEMIA

```
Código      Nome       Altura      Peso
=====
1          John        1.8         84.0
2          Jane        1.85        70.0
3          Carol       1.5         58.0
Média:           1.72        70.67
=====
```

Downey, Allen B.. Pense em Python (p. 162). Novatec Editora. Edição do Kindle.