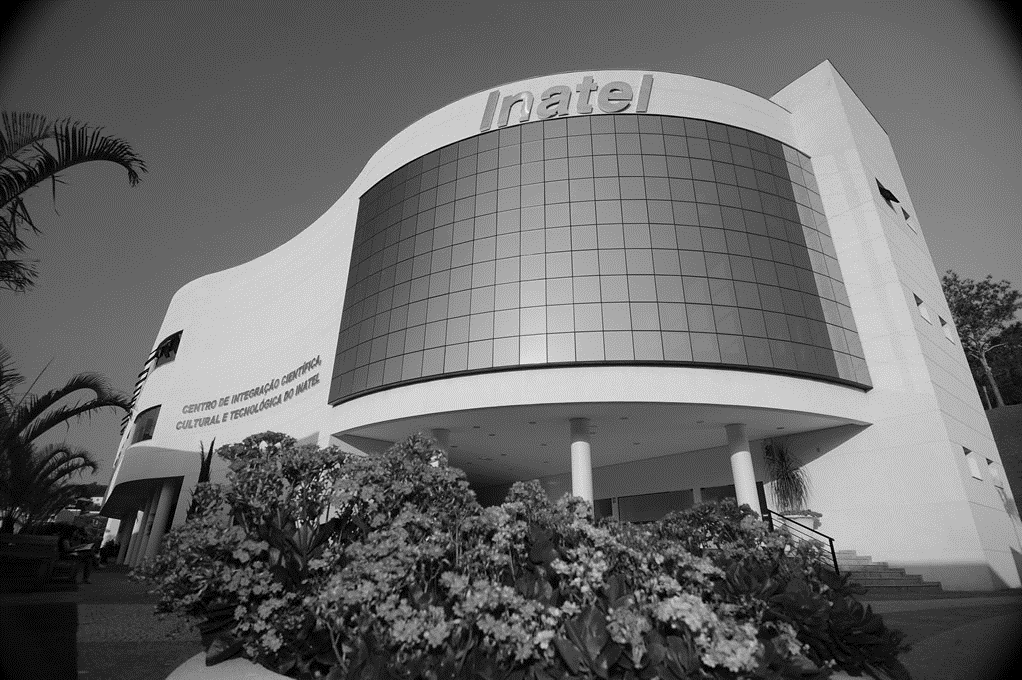
**Instituto Nacional de Telecomunicações - Inatel**



**PYTHON**

**Guia básico sobre a linguagem de programação Python**

Time Core MIDC – Versão 1– fevereiro de 2023

Sumário

[Controle de Versões 3](#_Toc129773265)

[Breve resumo 3](#_Toc129773266)

[Python 3](#_Toc129773267)

[Breve História 3](#_Toc129773268)

[Versão 4](#_Toc129773269)

[Download Python 4](#_Toc129773270)

[Introdução ao PyCharm 4](#_Toc129773271)

[IDE 4](#_Toc129773272)

[PyCharm 4](#_Toc129773273)

[Download PyCharm 4](#_Toc129773274)

[Criando um projeto no PyCharm 4](#_Toc129773275)

[Criando uma classe no PyCharm 6](#_Toc129773276)

[**Estrutura de dados 6**](#_Toc129773277)

[Sequencia 6](#_Toc129773278)

[Listas 7](#_Toc129773279)

[Tuplas 8](#_Toc129773280)

[Range 8](#_Toc129773281)

[Variáveis 9](#_Toc129773282)

[Nomes das variáveis 9](#_Toc129773283)

[Operadores aritméticos 10](#_Toc129773284)

[Operadores de comparação 10](#_Toc129773285)

[Operadores lógicos 10](#_Toc129773286)

[Strings 11](#_Toc129773287)

[Entrada de dados 11](#_Toc129773288)

[Saída de dados 12](#_Toc129773289)

[Comandos IF, ELSE e ELIF 12](#_Toc129773290)

[IF 12](#_Toc129773291)

[ELSE 12](#_Toc129773292)

[ELIF 13](#_Toc129773293)

[Comando while 13](#_Toc129773294)

[Comando for 13](#_Toc129773295)

[Funções 14](#_Toc129773296)

[O que é uma função? 14](#_Toc129773297)

[Parâmetros da função 15](#_Toc129773298)

[Função com retorno 16](#_Toc129773299)

[Retornando múltiplos valores 17](#_Toc129773300)

[Número arbitrário de parâmetro (\*args) 17](#_Toc129773301)

[Número arbitrário de chave (\*\*kwargs) 18](#_Toc129773302)

[Arquivos 18](#_Toc129773303)

[Escrita de um arquivo 18](#_Toc129773304)

[Fechando um arquivo 19](#_Toc129773305)

[Escrevendo palavras em novas linhas 19](#_Toc129773306)

[Orientação a objetos 19](#_Toc129773307)

[Classes e objetos 19](#_Toc129773308)

[Métodos 20](#_Toc129773309)

[Python é objeto 20](#_Toc129773310)

[Encapsulamento 20](#_Toc129773311)

# Controle de Versões

Controle de versões, informando o autor responsável pela atualização, e o conteúdo adicionado ao documento.

|  |  |  |  |
| --- | --- | --- | --- |
| Versão | Autor | Data | Conteúdo atualizado ou adicionado. |
| 1.0 | Time Core | 09/03/2023 | Versão inicial. |

# Breve resumo

O presente documento, visa apresentar o entendimento sobre a linguagem de programação Python de uma forma simples e prática, tendo em vista ela, sendo como orientada a objetos.

## Python

Python é uma linguagem de programação orientada a objetos de alto nível semântico. Sua simplicidade beneficia usuários de todo o mundo e auxilia a manutenção de determinados programas.

É uma das linguagens mais populares para a análise de dados. Aplicativos como *Netflix, Spotify* e *DropBox* são escritos em Python. Sua capacidade é bastante ampla, sendo assim, uma das linguagens que mais tem crescido ao longo dos anos devido a sua compatibilidade.

### Breve História

A linguagem de programação Python foi criada em 1990 por Guido Van Rossum no Instituto de Pesquisa Nacional para Matemática e Ciência da Computação (CWI) na Holanda como sucessora da linguagem ABC.

A linguagem ABC era mais produtiva que as demais linguagens, mas ela não possuía funcionalidades importantes para a interação com o sistema operacional e um dos focos principais do Python era aumentar a produtividade do programador.

Guido se desafiou a criar uma linguagem de script simples, a fim de melhorar a linguagem ABC. Uso de dicionários, declarações básicas e uso obrigatório de indentação são conceitos que diferenciam Python da linguagem ABC.

Em maio de 2000, Guido e seu time de Python se mudaram para BeOpen para formar o PythonLabs. Em outubro de 2000, o time da PythonLabs se moveu para a Digital Creations. Em 2001, a Python Software Foundation foi formada especialmente para manter a linguagem e hoje possui sua propriedade intelectual.

### Versão

Há diversas versões de Python para o download, porém o Python3 é mais maduro e mais recomendável para uso. No entanto, para sistemas operacionais como Linux e Mac OS o Python2 já vem pré-instalado.

### Download Python

Como mencionado anteriormente, para sistemas como Linux e Mac OS o Python já vem instalado, porém para sistemas como Windows é necessário fazer o download pelo site <https://www.python.org/> , além de algumas configurações extras.

### Introdução ao PyCharm

### IDE

Para adentramos em códigos que requerem maior complexidade será utilizado um Ambiente Integrado de Desenvolvimento ou IDE ((sigla em inglês para Integrated Development Enviroment). Uma IDE é um software com funcionalidades que auxiliam no desenvolvimento do código.

### PyCharm

O PyCharm é uma multiplataforma com versões para diversos sistemas operacionais. É desenvolvido pela empresa JetBrains e fornece depurador gráfico, autocompletamento de código, capacidades de navegação e etc.

Com o PyCharm é possível desenvolver em Python, porém também suporta HTML, CSS, JavaScript e XML.

### Download PyCharm

É necessário fazer o download do PyCharm pelo site <https://www.jetbrains.com/pycharm/> .

### Criando um projeto no PyCharm

Ao clicar no ícone “File” na tela inicial do PyCharm, aparecerá diversas informações. Deverá ser selecionado “New Project”.

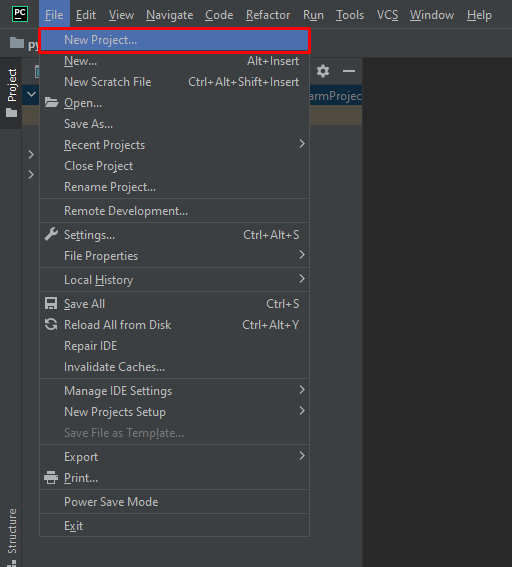


Figura 1

É na janela de “New Project” que será feito todas as configurações necessárias:

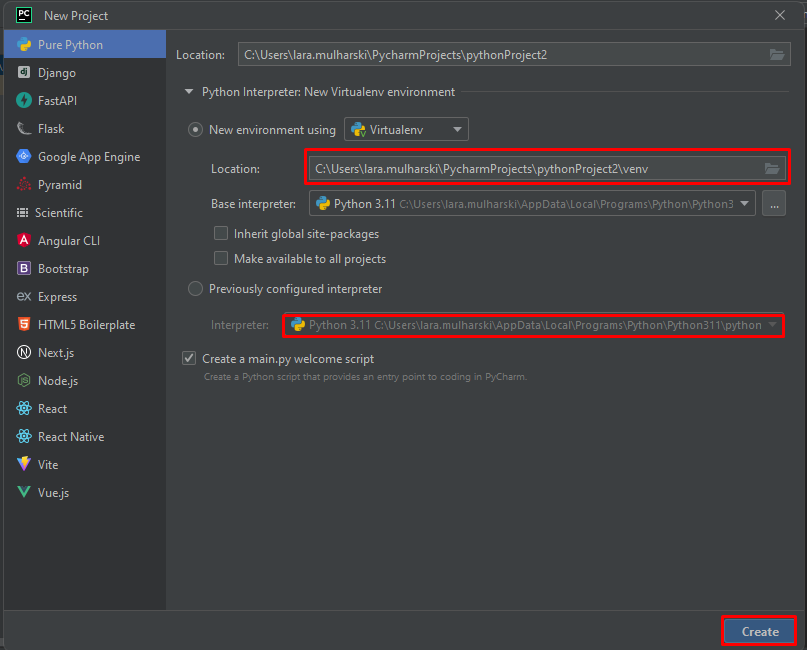


Figura 2

Primeiramente é necessário especificar o nome do projeto. O PyCharm sugere um local padrão para salvar o projeto, ao selecionar o caminho padrão a IDE vai criar uma pasta chamada PyCharmProjects na pasta home do dispositivo. Após isso, deve-se selecionar a versão do interpretador Python. Por fim, selecionar Create e o projeto é criado.

### Criando uma classe no PyCharm

Para criar uma classe no PyCharm é simples, basta clicar com o botão direito na pasta “Venv” que foi criada automaticamente com o projeto, selecionar “New’ e logo em seguida “Python File”.

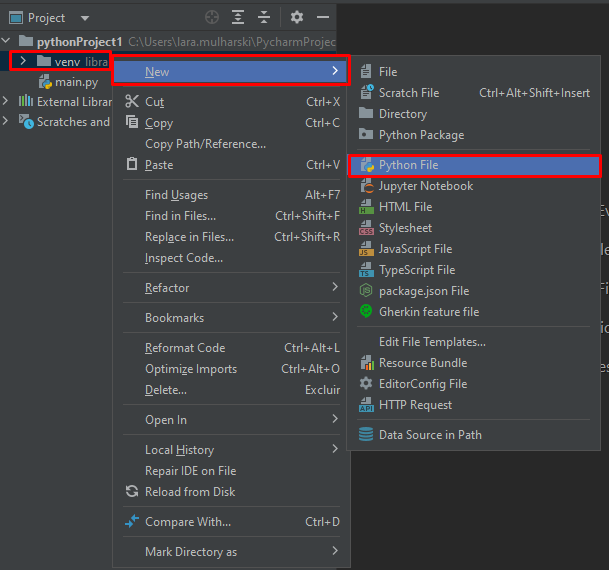


Figura 3

# Estrutura de dados

## Sequencia

O Python possui sequência de dados e será apresentado alguma delas. Existem três tipos básicos de sequência: list (lista), tuple (tupla) e range (objeto de intervalo).

Como por exemplo, existe também a string, que é uma sequência de texto que já foi apresentada anteriormente.

As sequências são classificadas em mutáveis e imutáveis. Tuplas, string e ranges são imutáveis, enquanto a lista é mutável

Segue a tabela com algum dos tipos de sequências. Para maior entendimento, **s** e **t** são sequencias do mesmo tipo, **n**, **i**, **j** e **k** são inteiros e **x** é um objeto arbitrário que atende a qualquer tipo e restrição de valor imposto por s

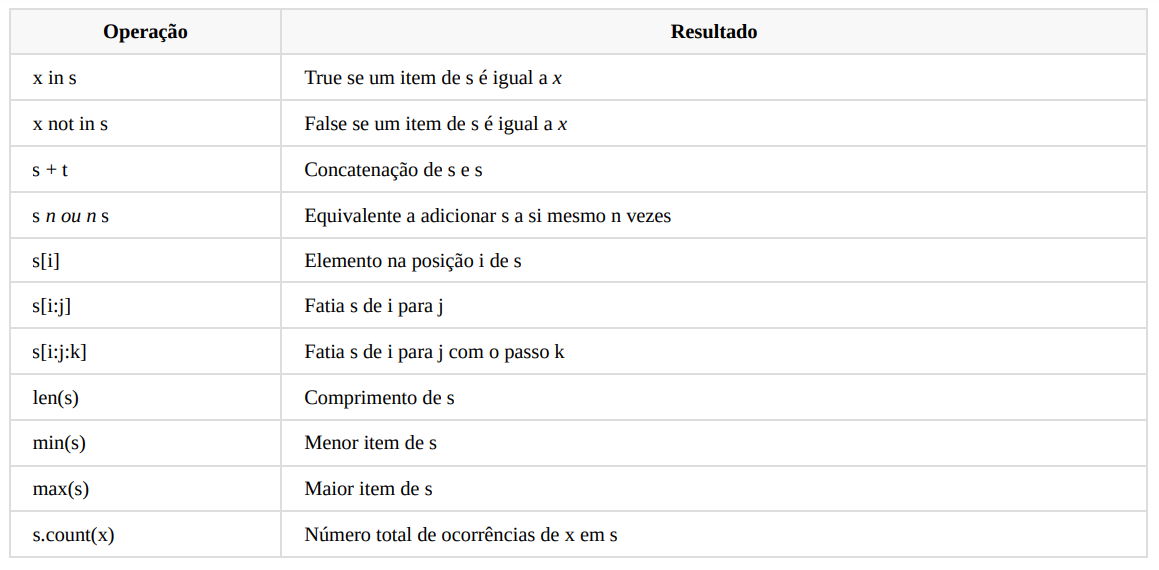


Figura 42

## Listas

Lista é uma sequência de valores e cada valor é indicado por um índice que inicia em 0. Os elementos de uma lista podem ser de qualquer tipo, são delimitados por [ ] e seus elementos são separados por vírgula. Lembrando que o valores dos elementos de uma lista podem ser alterados.

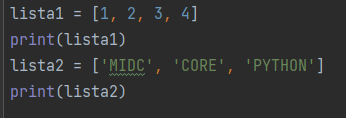


Figura 43

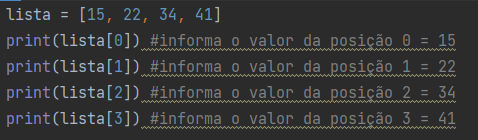


Figura 44

É possível acessar múltiplos valores através do fatiamento. **Lembrando que a primeira posição dos elementos inicia em “0”**. Segue o exemplo:

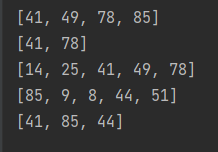


Figura 45

## Tuplas

As tuplas são imutáveis, não sendo possível realizar a alteração dos seus elementos depois de criados. Diferentemente da lista, o delimitador de uma tupla é o ( )



Figura 46

Outro modo:

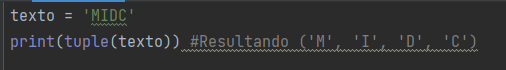


Figura 47

## Range

O range é uma sequência imutável de números, usado normalmente para looping de uma quantidade de vezes no comando for obedecendo a seguinte sintaxe:

Range (início, fim)



Figura 48

O range apenas armazena seu início e fim. Para imprimir seus valores, é necessário o utilizar o *for*

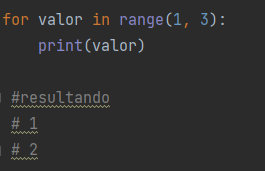


Figura 49

### Variáveis

As variáveis são responsáveis por armazenar as informações que serão utilizadas em algum momento no programa. Elas podem ser alteradas e serem utilizadas para cálculos, controle de estado e exibição de conteúdo. Para criar uma variável, basta usar o sinal de igualdade ( = ) e atribuir um valor a ela:

Os tipos de variáveis no Python são:

* **Inteiro (int)** à É o tipo usado para um número inteiro, podendo ter ou não sinal, ou seja, ser positivo ou negativo. Por exemplo: 1, 0, e −5
* **Ponto Flutuante (float)** à O ponto flutuante é o tipo usado para números racionais (números que podem ser representados por uma fração) informalmente conhecido como “número quebrado”. Por exemplo: 1.78, 2.5 e 8.6
* **Complexo (complex)** à É o tipo de dado usado para representar números complexos. Um tipo complexo possui duas partes: a parte real e a parte imaginária.
* **String (str)**  à É o tipo geralmente utilizado para representar palavras, frases ou textos.
* **Boolean (bool)** à Tipo de dado que pode assumir apenas dois valores: falso ou verdadeiro (False ou True). Podem ser considerados como 0 ou 1.
* **List (list)** à Listas agrupam um conjunto de elementos variados, podendo conter: inteiros, floats, strings, outras listas e outros tipos. Elas são definidas utilizando-se colchetes para delimitar a lista e vírgulas para separar os elementos.
* **Tuple** à Da mesma forma que a Lista, Tupla é um tipo que agrupa um conjunto de elementos. Mas sua forma de definição é diferente, é utilizado parênteses e também separado por vírgula. A diferença para Lista é que Tuplas são imutáveis, ou seja, após sua definição, Tuplas não podem ser modificadas.
* **Dictionary (dic)** à Tipo que é utilizado para agrupar elementos através da estrutura de chave e valor, onde a chave é o primeiro elemento seguido por dois pontos e pelo valor.

Para verificar o tipo da variável, utiliza-se a função **type().**

### Nomes das variáveis

 As variáveis podem ser nomeadas de acordo com a vontade do programador, podendo conter letras e números. As variáveis devem começar com letra minúscula e não podem ser nomeadas com palavras reservadas da linguagem. As palavras reservadas são:

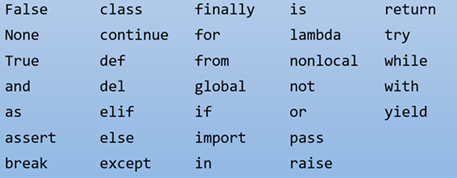


Figura 4

### Operadores aritméticos

Operadores aritméticos são símbolos especiais que representam cálculos. Para fazer cálculos com números utilizamos os operadores +, -, \*, /, \*\*, // e % que representam, respectivamente, adição, subtração, multiplicação, divisão, potenciação, divisão inteira e resto da divisão.

Quando mais de um operador aparece em uma expressão, a ordem em que são realizadas as operações dependem das regras de precedência. A ordem é:

1. ()
2. \*\*
3. \* / // %
4. + -

### Operadores de comparação

Operadores de comparação são utilizados para comparar valores. Eles irão retornar True ou False, dependendo da condição. Com isso, é possível criar condições para os códigos. A tabela a seguir mostra exemplos usados em Python.

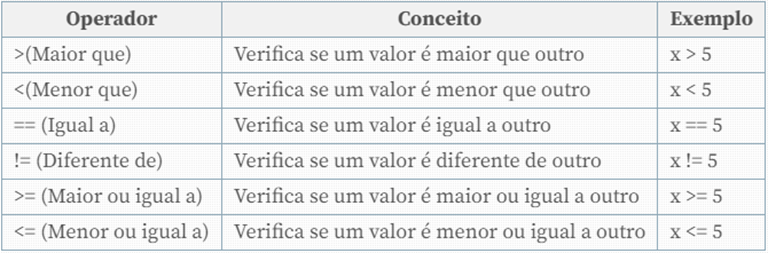


Figura 5

### Operadores lógicos

Operadores lógicos são empregados para conectar duas ou mais expressões condicionais. A tabela a seguir mostra exemplos usados em Python.

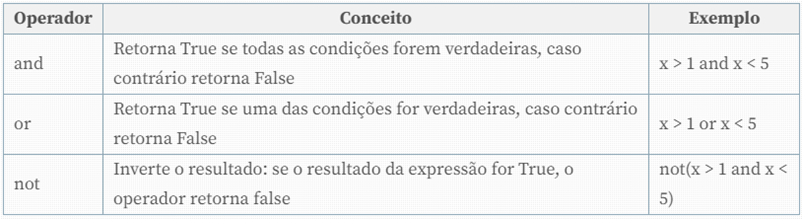


Figura 6

### Strings

Strings apresentam muitas utilidades prontas, que são chamadas de métodos. Como por exemplo, o método **isalpha( )** verifica se a entrada é alfabética, **islower( )** verifica se  entrada é minúscula. Há outras funcionalidades de strings que podem ser acessadas neste link: [https://docs.python.org/3/library/stdtypes.html#string-methods](https://docs.python.org/3/library/stdtypes.html%23string-methods)



Figura 7



Figura 8

### Entrada de dados

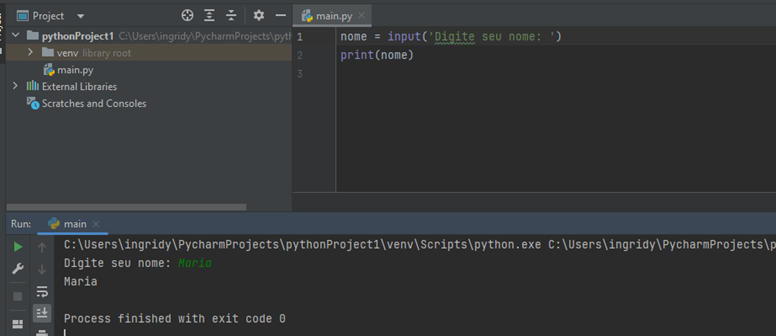
Geralmente é necessário interagir com o usuário para obter dados. O Python possui uma função que captura a entrada de valores, a função é a **input()**. Quando a função é chamada, o programa para e espera o usuário digitar algo, e apertar a tecla ENTER. Após a entrada de dados pelo usuário, o programa irá continuar sua execução, passando para as instruções seguintes.

Figura 9

### 

### Saída de dados

A função para imprimir dados em Python, é a função **print( )**. Ela é responsável por mostrar valores em seu terminal. Há duas formas de usar a função print:

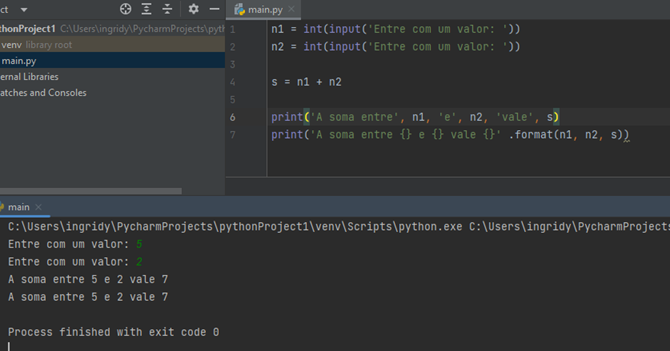


Figura 10

A opção de imprimir concatenando a string com a variável e usando a função **format(** **)**. A função **format( )** é mais recomendada, pois facilita a impressão e formatação dos dados. Ela vai substituir o  {} pela variável.

### Comandos IF, ELSE e ELIF

### IF

O if é uma estrutura de condição que avalia uma expressão, e executa uma ação de acordo com o resultado.

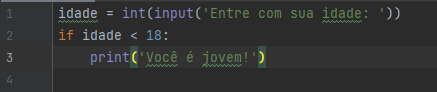


Figura 11

### ELSE

Como visto no exemplo anterior, caso a condição IF não seja atendida, nenhum resultado foi definido. Com isso é necessário utilizar o comando else, que irá ser executado se a condição anterior não for satisfeita.

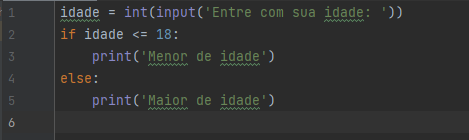


Figura 12

### ELIF

Se existir mais de uma condição que necessite ser analisada, é utilizado o comando elif.

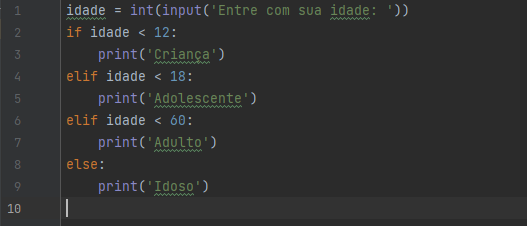


Figura 13

### Comando while

O comando while é uma estrutura de repetição que permite executar um bloco de código enquanto uma condição for verdadeira. Para utilizar o while, é necessária uma expressão de controle que utiliza uma variável de apoio para indicar o ponto de parada.

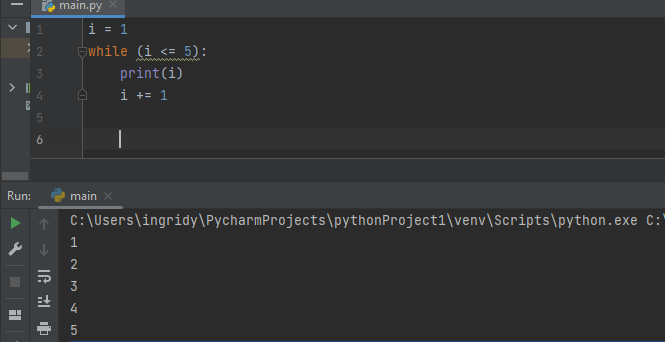


Figura 14

Como visto, a variável “i” funciona como um contador e é utilizada para controlar a repetição de loop.

### Comando for

O comando for também é uma estrutura de repetição assim como o while. A diferença é que ele executa a instrução um certo número de vezes, e o while executa até que a condição seja atendida.

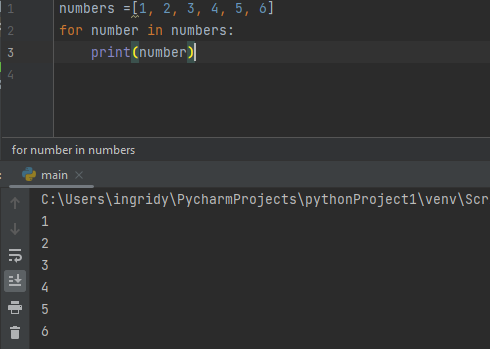


Figura 15

### Funções

### O que é uma função?

Na computação, função é uma sequência de instruções condensadas podendo ou não receber parâmetros externos. Como na maioria das linguagens de programação, existem funções já pré-programadas como as vistas no capítulo anterior, como o **print ()**, mas podemos também criar nossas próprias funções, com nossos próprios parâmetros e instruções.

Por exemplo, podemos criar uma função que faz o cálculo da média das notas dos alunos de uma turma:

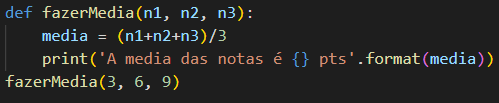


Figura 12

A função recebe os parâmetros **n1, n2 e n3** que são os valores das notas, dentro da função é definida a variável média que recebe o valor da operação da divisão das três notas por três, e por fim é chamada a função print que mostra o resultado na tela.

Existem também funções sem parâmetros, por exemplo essa função que somente mostra algo na tela:

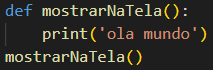


Figura 13

### Parâmetros da função

Parâmetros consistem em uma lista com zero ou mais elementos, podendo ser obrigatórios ou opcionais. Para este parâmetro ser opcional, é necessário atribuir um valor padrão a ele (default). O valor mais usual para esses parâmetros é o **None**. Por exemplo:

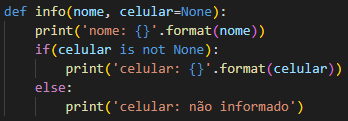


Figura 14

A função acima recebe um número de telefone e verifica com o bloco **if/else** se o valor é igual ou diferente de **None**, se for diferente ela mostra o valor informado para a variável **celular** e se for igual ela mostra que o celular não foi informado.

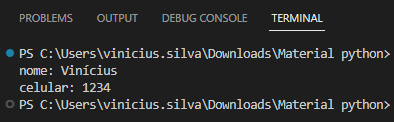


Figura 15

Figura 16

Com a entrada de dados **(“Vinícius”, 1234)** a função tem como saída o nome e o celular. Agora, se tivermos uma entrada somente com o nome a saída mostra o nome e que o celular não foi informado.





Figura 17

Figura 18

Agora, se entrarmos somente com o valor do número de telefone, a saída mostrará o celular como o nome, já que não foi informado um valor para o parâmetro celular na estrutura de entrada.



Figura 19

Figura 20

Com isso, percebemos que o Python interpreta os dados na ordem em que são passados, por isso ele interpretou o número de celular e o atribuiu como valor da variável **nome**. Para resolvermos isso, devemos informar qual parâmetro receberá um valor nulo.

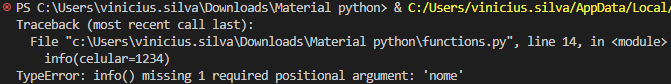


Figura 21

Como não existe um caso no qual o parâmetro nome não é informado, a função retorna um erro, mas o valor informado é atribuído ao parâmetro **celular**.

### Função com retorno

Dependendo do caso, não queremos que uma função mostre o valor diretamente na tela, por isso podemos fazer com que a função apenas retorne um valor. Podemos imprimir esse valor desta forma:

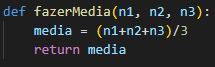


Figura 22



Figura 23

Usando este método, o terminal mostrará somente o valor do resultado da média entre os 3 valores que foram recebidos pela função. Sendo assim, podemos também atribuir o valor do resultado retornado pela função a uma variável e assim mostrar na tela.



Figura 24

Agora podemos usar o valor da variável que recebeu o resultado da função para, por exemplo, calcular a média final de um aluno.

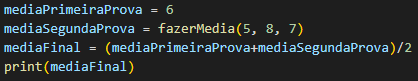


Figura 25

Uma função pode ter mais de um chamado de **return**, mas terá apenas um retorno. A seguir, por exemplo, a função vai retornar o nome e o número de celular caso o parâmetro **celular** não seja nulo, ou retornar o nome e mostrar que o celular não foi informado caso o parâmetro **celular** seja nulo.

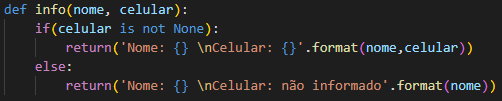


Figura 26

### Retornando múltiplos valores

Apesar de uma função executar somente uma vez o retorno, como explicado acima, podemos retornar mais de um valor em um único retorno. Vamos montar uma função que retorna o resultado da multiplicação e divisão entre dois números.

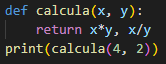


Figura 27

Essa função retorna um dado do tipo **tuple**, que funciona como um **array** ordenado e que não pode ser alterado. Sendo assim, podemos fazer um bloco **for** para imprimir o retorno desse tipo de função.

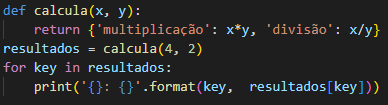


Figura 28

### Número arbitrário de parâmetro (\*args)

Em Python podemos passar um número arbitrário de argumentos para uma função, para isso utilizamos os parâmetros **\*args** e **\*\*kwargs**. Não é necessário utilizar especificamente os nomes **\*args** e **\*\*kwargs**, podemos usar apenas **\*** ou **\*\***. Podemos escrever, por exemplo, **\*var** e **\*\*vars**.

Primeiro vamos aprender a usar o **\*args**, ele pode ser usado, assim como o **\*\*kwargs**, em definições de funções. Esses parâmetros nos permitem passar um número variável de argumentos a uma função. O parâmetro **\*args** serve para quando não se sabe previamente o número de argumentos que serão passados para a função.

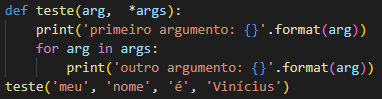


Figura 29

Essa função gera a seguinte saída:

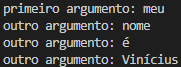


Figura 30

O parâmetro **arg** funciona como qualquer outro parâmetro, já o **\*args** recebe múltiplos valores dentro de algum tipo de **array**, podendo ser usados também listas do tipo **tuple** e **list**.

O parâmetro **\*args**, em suma, é utilizado quando não sabemos com antecedência o número de argumentos que serão passados para a função. O asterisco faz um empacotamento dos dados para ajudar e facilitar a disponibilização dos parâmetros, e a função que recebe esse tipo de parâmetro faz o desempacotamento.

### Número arbitrário de chave (\*\*kwargs)

O **\*\*kwargs** permite que sejam passados um tamanho variável da palavra-chave dos argumentos para uma função. Para isso é preciso utilizar o **\*\*kwargs** para manipular argumentos nomeados em uma função.

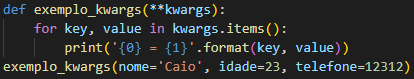


Figura 31

A diferença entre **\*args** e **\*kwargs** é que **\*args** espera um argumento do tipo **tuple** e **\*\*kwargs** espera entradas do tipo **dictionary**, com argumentos nomeados.

### Arquivos

### Escrita de um arquivo

Para abrir um arquivo, o Python possui uma função *open()* . Essa função recebe dois parâmetros: o primeiro é o nome do arquivo a ser aberto, e o segundo parâmetro é o modo que é trabalhado esse arquivo (ler ou escrever). O modo é expresso através de uma string: “w” sendo write (escrita) e “r” sendo read (leitura).

Exemplo do comando no terminal PyCharm:



Figura 32

O arquivo criado se chama ‘palavras.txt’ e está no modo escrita. É essencial saber que o modo de escrita sobrescreve o arquivo. Caso a intenção seja apenas adicionar algum tipo de conteúdo ao arquivo é utilizado o modo “a” (abreviação de *append*).

Para escrever algum conteúdo no arquivo basta utilizar a função *write ()* dentro do arquivo, sendo assim:



Figura 33

O retorno dessa função será o número de caracteres de cada texto do arquivo, ou seja, o retorno da função acima deverá ser 4.

### Fechando um arquivo

Quando se trabalha com um arquivo, é importante fechá-lo. Para fechá-lo é usado a função *close ().*



Figura 34

### Escrevendo palavras em novas linhas

Para escrever palavras em outra linha é simples, basta adicionar \n ao final da palavra:



Figura 35

### Orientação a objetos

### Classes e objetos

Quando é preparado um bolo, geralmente há uma receita que define os ingredientes e o modo de preparação. No mundo OO essa “receita” recebe o nome de **classe**. Ou seja, antes de criar um objeto é necessária uma classe.

Outra analogia interessante para se fazer é com a construção de uma casa. O primeiro ponto é necessário a planta da casa para ela ser construída e depois, a casa em si. Ou seja, o projeto da casa é a classe e a casa construída é o objeto.

Exemplo do código:

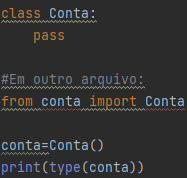


Figura 36

### Métodos

No paradigma orientado a objetos, as funcionalidades de um objeto são chamadas de **métodos** (funções dentro de uma classe).

### Python é objeto

Quando se menciona a linguagem de programação Python é inevitável pensar que ela é totalmente orientada a objeto, tudo em Python é objeto! A palavra class aparece quando é solicitado ao Python para devolver o tipo de uma variável através da função type.

Afinal, um sistema orientado a objetos é um grande conjunto de classes, delegando responsabilidades para quem for mais apto a realizar uma determinada tarefa.

### Encapsulamento

Encapsular é fundamental para que o sistema seja suscetível a mudanças. O conjunto de métodos públicos de uma classe é também chamado de interface da classe, pois é a única maneira a qual você se comunica com objetos da classe.

Para permitir o acesso aos atributos de uma maneira controlada, é mais comum criar dois métodos, um que retorna e outro que muda o valor. A convenção para esses métodos em muitas linguagens orientadas a objetos é colocar a palavra ***get*** ou ***set*** antes do nome do atributo.

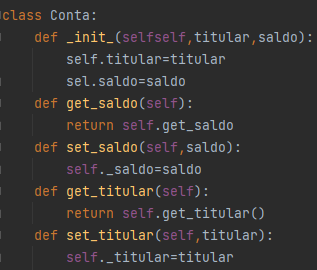


Figura 37

***Getters*** e ***setters*** são usados em muitas linguagens de programação orientada a objetos para garantir o princípio do encapsulamento de dados. O encapsulamento de dados é visto como agrupamento de dados com métodos que operam nesses dados. Esses métodos são, os getters para recuperar dados e os setters para alterar os dados.