**CONHECIMENTOS P/ PYTHON**

Organizar a ideia melhor sobre como colocar esses comandos

1. **Sumário**

[1. Introdução / Objetivo 4](#__RefHeading___Toc903_1204716596)

[1.1. Ambientes Virtuais Python 4](#__RefHeading___Toc3679_301590980)

[1.1.1. Criando o ambiente virtual com venv 4](#__RefHeading___Toc3681_301590980)

[2. Funções 5](#__RefHeading___Toc2248_4049308617)

[2.1. Chamando uma função 5](#__RefHeading___Toc2250_4049308617)

[2.2. Como acrescentar novas funções 5](#__RefHeading___Toc2252_4049308617)

[2.3. Parâmetros e argumentos 6](#__RefHeading___Toc2254_4049308617)

[2.4. Escopo Local e Global 6](#__RefHeading___Toc2575_2911162571)

[2.5. Função Lambda 6](#__RefHeading___Toc2577_2911162571)

[3. Métodos Comuns para Strings 6](#__RefHeading___Toc25_1549371939)

[3.1. Método len(): 6](#__RefHeading___Toc35_1549371939)

[3.2. Métodos upper() e lower(): 7](#__RefHeading___Toc37_1549371939)

[3.3. Método isalpha(): 7](#__RefHeading___Toc39_1549371939)

[3.4. Método isnumeric(): 7](#__RefHeading___Toc62_1549371939)

[3.5. Método strip(): 8](#__RefHeading___Toc41_1549371939)

[3.6. Método join(): 8](#__RefHeading___Toc64_1549371939)

[3.7. Métodos Split 8](#__RefHeading___Toc27_1549371939)

[3.8. Método capitalize(): 9](#__RefHeading___Toc66_1549371939)

[3.9. Método count(): 9](#__RefHeading___Toc68_1549371939)

[3.10. Método replace(): 9](#__RefHeading___Toc70_1549371939)

[3.11. Método find(): 10](#__RefHeading___Toc72_1549371939)

[3.12. Método title(): 10](#__RefHeading___Toc74_1549371939)

[4. Manipulação de Strings 10](#__RefHeading___Toc2587_2911162571)

[4.1. Caracteres de escape 10](#__RefHeading___Toc2589_2911162571)

[4.2. Strings de múltiplas linhas com aspas triplas 11](#__RefHeading___Toc2591_2911162571)

[5. F-String: 11](#__RefHeading___Toc76_1549371939)

[5.1. F-string e suas formatações: 11](#__RefHeading___Toc285_1934664425)

[5.2. Método round() no f-string 12](#__RefHeading___Toc358_1368987719)

[5.3. Alinhamento a Esquerda, Direita e Centro: 12](#__RefHeading___Toc78_1549371939)

[6. Fatiamento de Strings 13](#__RefHeading___Toc309_1368987719)

[6.1. Como fatiar em python 14](#__RefHeading___Toc311_1368987719)

[6.2. Percorrer uma fatia com um laço 14](#__RefHeading___Toc608_3488376286)

[7. Utilizações em Listas 15](#__RefHeading___Toc321_1368987719)

[7.1. Criando uma lista 15](#__RefHeading___Toc2388_2600184081)

[7.2. Acessando elementos em uma lista 16](#__RefHeading___Toc2390_2600184081)

[7.3. Adicionando elementos na lista (*append e insert*): 17](#__RefHeading___Toc2163_2842804153)

[7.4. Juntando duas listas (*extend*): 17](#__RefHeading___Toc2165_2842804153)

[7.5. Excluindo elementos de uma lista: 18](#__RefHeading___Toc2392_2600184081)

[7.6. Como saber se um elemento está na lista? 18](#__RefHeading___Toc2394_2600184081)

[7.7. Método Sort() 19](#__RefHeading___Toc323_1368987719)

[7.8. Copiando uma lista 19](#__RefHeading___Toc610_3488376286)

[7.9. Transferindo itens de uma lista para outra: 20](#__RefHeading___Toc693_2599393118)

[7.10. Removendo instâncias de uma lista: 21](#__RefHeading___Toc695_2599393118)

[7.10.1. Com *remove()* 21](#__RefHeading___Toc2167_2842804153)

[7.10.2. Com pop(): 21](#__RefHeading___Toc2169_2842804153)

[7.11. Passando uma lista para uma função 21](#__RefHeading___Toc869_1204716596)

[7.12. Descompactação de Zip (listas) 22](#__RefHeading___Toc2256_4049308617)

[7.13. Trabalhando com Listas 22](#__RefHeading___Toc2579_2911162571)

[7.13.1. Utilizando loops for com listas 22](#__RefHeading___Toc2581_2911162571)

[7.13.2. Operadores IN e NOT IN 23](#__RefHeading___Toc2583_2911162571)

[8. Utilizações em Tuplas 23](#__RefHeading___Toc325_1368987719)

[9. Utilizações em Dicionários 24](#__RefHeading___Toc327_1368987719)

[9.1. Criando um dicionário 24](#__RefHeading___Toc2313_146788991)

[9.2. Adicionando elementos em um dicionário já criado 25](#__RefHeading___Toc2315_146788991)

[9.3. Modificando valores em um dicionário 25](#__RefHeading___Toc670_2599393118)

[9.4. Verificando se um valor e chave estão no dicionário (get) 25](#__RefHeading___Toc2585_2911162571)

[9.5. Percorrendo um dicionário com um laço 26](#__RefHeading___Toc672_2599393118)

[9.5.1. Percorrendo todas as chaves com um laço 26](#__RefHeading___Toc674_2599393118)

[9.5.2. Percorrendo todos os valores com um laço com values() e set() 27](#__RefHeading___Toc676_2599393118)

[9.6. Preenchendo um dicionário com input 27](#__RefHeading___Toc697_2599393118)

[9.7. Métodos úteis nos dicionários 28](#__RefHeading___Toc2317_146788991)

[10. Informações Aninhadas 29](#__RefHeading___Toc678_2599393118)

[10.1. Uma lista de dicionários 29](#__RefHeading___Toc680_2599393118)

[10.2. Uma lista em um dicionário 30](#__RefHeading___Toc682_2599393118)

[10.3. Um dicionário em um dicionário 30](#__RefHeading___Toc684_2599393118)

[11. Módulos em Python 30](#__RefHeading___Toc360_1368987719)

[11.1. Importando uma função (própria) para um módulo 31](#__RefHeading___Toc871_1204716596)

[11.2. Módulo datetime (*para datas e horas*) 31](#__RefHeading___Toc3688_301590980)

[11.2.1. Problema do fuso horário 32](#__RefHeading___Toc3690_301590980)

[11.2.2. Calculando datas 33](#__RefHeading___Toc2517_1966780586)

[11.3. Módulo Calendar (*trabalhar com calendários*) 33](#__RefHeading___Toc3692_301590980)

[11.4. Módulo os 34](#__RefHeading___Toc1983_2166851155)

[11.5. Módulo Random 35](#__RefHeading___Toc362_1368987719)

[11.6. Módulo secrets 37](#__RefHeading___Toc2040_3334238204)

[11.7. Módulo subprocess 37](#__RefHeading___Toc2093_239100574)

[11.8. Módulo PyPDF2 38](#__RefHeading___Toc2111_239100574)

[11.9. Openpyxl – Trabalhando com arquivos Excel, xlsx, xlsm, xltx e xltm. 40](#__RefHeading___Toc2522_1966780586)

[11.10. Pillow – redimensionamento de imagens 40](#__RefHeading___Toc2524_1966780586)

[12. Cores em Python 40](#__RefHeading___Toc364_1368987719)

[13. Expressões Regulares em Python 41](#__RefHeading___Toc366_1368987719)

[14. Funções para validação de dados 41](#__RefHeading___Toc2247_3310496464)

[14.1. LeiaInt() 41](#__RefHeading___Toc2893_3310496464)

[14.2. valida\_int() 41](#__RefHeading___Toc3796_3310496464)

[14.3. Validar CPF 42](#__RefHeading___Toc1294_3595150828)

[15. Orientação a Objetos 42](#__RefHeading___Toc985_3267680264)

[15.1. O que é Programação Orientada a Objetos? 42](#__RefHeading___Toc987_3267680264)

[15.2. Classes 42](#__RefHeading___Toc886_1204716596)

[15.3. O Atributo 43](#__RefHeading___Toc989_3267680264)

[15.4. O Objeto 44](#__RefHeading___Toc991_3267680264)

[15.5. A Mensagem 44](#__RefHeading___Toc993_3267680264)

[15.6. Método 44](#__RefHeading___Toc995_3267680264)

[15.7. O que é UML? 45](#__RefHeading___Toc997_3267680264)

[15.8. Herança 45](#__RefHeading___Toc999_3267680264)

[15.9. Polimorfismo 46](#__RefHeading___Toc1001_3267680264)

[15.10. Definir uma classe em Python 46](#__RefHeading___Toc1003_3267680264)

[15.10.1. Introdução ao Método \_\_init\_\_ 47](#__RefHeading___Toc1005_3267680264)

[15.10.1.1. Método \_\_init\_\_() de uma subclasse (herança) 47](#__RefHeading___Toc1231_4204049732)

[15.10.2. Entendendo o self em classes 47](#__RefHeading___Toc1236_3267680264)

[15.11. Exemplos Práticos em Python 48](#__RefHeading___Toc1013_3267680264)

[15.11.1. Modificar valores em instâncias 50](#__RefHeading___Toc1233_4204049732)

[15.11.2. Modificando o valor de um atributo com um método 51](#__RefHeading___Toc1235_4204049732)

[15.11.3. Aprendendo a usar o método \_\_init\_\_ de uma classe filha 52](#__RefHeading___Toc1237_4204049732)

[15.11.4. Definindo atributos e métodos da classe-filha 53](#__RefHeading___Toc1239_4204049732)

[16. Arquivos 54](#__RefHeading___Toc1303_2970004907)

[16.1. Ler dados de um arquivo 54](#__RefHeading___Toc1305_2970004907)

[16.1.1. Lendo um arquivo inteiro 54](#__RefHeading___Toc1307_2970004907)

[16.1.2. Lendo dados linha a linha 55](#__RefHeading___Toc1309_2970004907)

[16.1.3. Criando uma lista de linhas de um arquivo 56](#__RefHeading___Toc1311_2970004907)

[16.2. Caminho de arquivo no Windows 56](#__RefHeading___Toc1313_2970004907)

[16.3. Modos de manipulação de arquivo 57](#__RefHeading___Toc1315_2970004907)

[16.4. Método úteis para arquivos 57](#__RefHeading___Toc1317_2970004907)

[16.4.1. Método write (e writelines) 58](#__RefHeading___Toc1319_2970004907)

[16.4.2. Método read, readline e readlines 58](#__RefHeading___Toc1321_2970004907)

[16.4.2.1. Read() 59](#__RefHeading___Toc1323_2970004907)

[16.4.2.2. Readline() x Readlines() 59](#__RefHeading___Toc1325_2970004907)

[16.4.3. Método seek 60](#__RefHeading___Toc1327_2970004907)

[16.4.4. Excluir arquivos 60](#__RefHeading___Toc1343_2970004907)

[16.4.5. Renomear arquivos 61](#__RefHeading___Toc1345_2970004907)

[16.4.6. Enconding 61](#__RefHeading___Toc1347_2970004907)

[17. Exceções 62](#__RefHeading___Toc1371_2970004907)

[17.1. Try, except, eles e finally 62](#__RefHeading___Toc1693_2656302054)

[17.2. Tratando exceções para arquivos 63](#__RefHeading___Toc1695_2656302054)

[17.2.1. FileNotFoundError 63](#__RefHeading___Toc1697_2656302054)

[17.2.2. Criando e lendo um arquivo com exceções 63](#__RefHeading___Toc1699_2656302054)

[18. Trabalhando com dados (tipos de dados) 64](#__RefHeading___Toc1701_2656302054)

[18.1. JSON – JavaScript Object Notation 64](#__RefHeading___Toc1985_2166851155)

[18.1.1. Utilizando json.dump() e json.load() 65](#__RefHeading___Toc1703_2656302054)

[18.1.2. Salvando um dado gerado por usuário 66](#__RefHeading___Toc1705_2656302054)

[18.2. CSV – Comma Separated Values 66](#__RefHeading___Toc1987_2166851155)

[18.2.1. Como ler o CSV no Python 67](#__RefHeading___Toc1994_2166851155)

[18.2.2. Como escrever em arquivos CSV no Python 67](#__RefHeading___Toc1996_2166851155)

[Referências 68](#__RefHeading___Toc18_1549371939)

# Introdução / Objetivo

Aqui vou fazer uma introdução para com esse documento e o porquê de eu ter criado ele.

Antes de falar de programas, comandos, métodos e muitas outras coisas, precisamos ver algumas coisas básicas sobre o Visual Studio Code, sobre o Python e algumas outras coisas.

## Ambientes Virtuais Python

Quando comecei a ‘brincar’ com a linguagem Python, eu aprendi sobre o ambiente virtual (ou venv como muitos conhecem) que tem a missão de carregar toda a sua instalação do Python para uma pasta no caminho escolhido. Geralmente todas as instalações e bibliotecas instaladas (seja com o pip ou com outros meios) vão para a pasta do venv.

Ao ativar um ambiente virtual, a instalação do ambiente virtual será usada. Venv é o módulo que é mais conhecido para criar ambientes virtuais.

Você pode dar o nome que preferir para um ambiente virtual, mas os mais comuns são: *venv – env - .venv - .env*.

### Criando o ambiente virtual com venv

Para criar uma venv é bem simples, basta ir no terminal, seja o do VSC ou do Pycharm, e digitar: **python -m venv <*nome.pasta*>**

O nome pasta tende a ser comumente chamado de venv ou de env.

Criar a pasta diretamente no terminal em que está localizado o projeto é de extrema facilidade pois a pasta vai direto para a pasta do projeto ao qual você está trabalhando, sem ter que movê-la ou recriá-la depois.

Para ativar o venv basta apenas digitar: **.\<*nome.pasta*>\Scripts\activate**

Para desativar é: **<*nome.pasta*>\Scripts\deactivate**

Esse “Scripts” vai mudar para “bin” se for no Mac ou Linux

# Funções

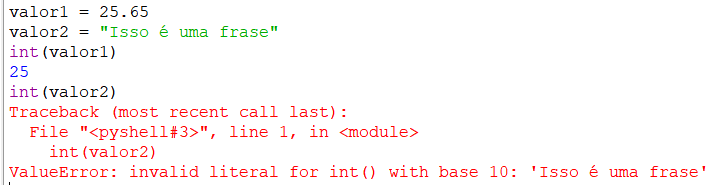
Uma função em python, nada mais é, do que uma sequência nomeada de instruções que executa uma operação de computação. Ao definir uma função, você especifica o nome e a sequência de instruções. Depois, pode “chamar” a função pelo nome.

Em python, algumas funções já vem pronta, já vem embutida na própria linguagem.

## Chamando uma função

Para chamar uma função, usamos seu nome e os parâmetros necessários. O Python oferece diversas funções, dentro elas existem as capazes de converterem valores de um tipo em outro.

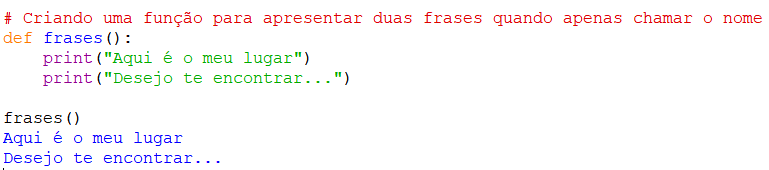
Exemplo é a função **int**, que recebe qualquer valor e o converte em um número inteiro, se for possível, ou declara que há um erro.

****

Existem outras funções que convertem outros valores, como, por exemplo, **float**, que converte um número em ponto flutuante,e **str**, que converte qualquer valor em string.

## Como acrescentar novas funções

Por enquanto, só usamos funções que vêm com o Python, mas também é possível acrescentar novas funções. Uma definição de função específica o nome de uma nova função e a sequência de instruções que são executadas quando a função é chamada.

**def** é uma palavra-chave que indica uma definição de função;

O nome da função é **frases**, e vale a observação que, as regras para nomes de funções são as mesmas que as das variáveis: letras, números e sublinhado, e o primeiro caractere não pode ser um número. Não podemos usar uma palavra-chave como nome de uma função e devemos evitar ter uma variável e uma função com o mesmo nome.

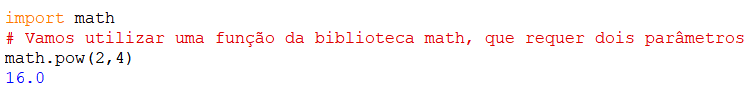
Os parênteses vazios depois do nome indicam que esta função não usa argumentos.

A primeira linha da definição de função chama-se cabeçalho, e ele precisa terminar em dois pontos;  
O resto é chamado de corpo. E precisa ser endentado, por convenção, a endentação é sempre de quatro espaços.

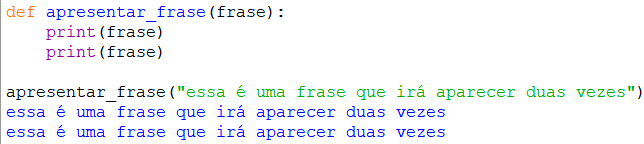
## Parâmetros e argumentos

A depender de quando você for criar suas funções, ou utilizar alguma já pronta do Python, exigiram argumentos.

Existe uma função matemática no Python que vamos utilizar para exemplo.

****

Vamos agora criar uma função e nela atribuiremos argumentos para ela:

****

## Escopo Local e Global

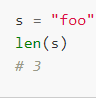
## Função Lambda

Aula 133 e 134 do curso que fala sobre a função lambda

# Métodos Comuns para Strings

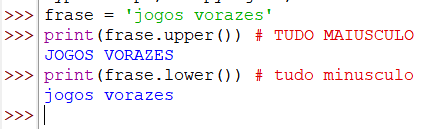
Um método em Python é uma função que pertence a uma classe ou a um objeto. Ele é utilizado para agrupar um conjunto de instruções relacionadas em um único bloco de código, permitindo sua reutilização em diferentes partes do programa.

## Método len():

O comando len() vai servir para mostrar o tamanho da string. Ele é uma função interna do Python e retorna o comprimento de um objeto, por exemplo, ele pode retornar o número de itens em uma lista. Pode-se usar a função com muitos tipos de dados diferentes.

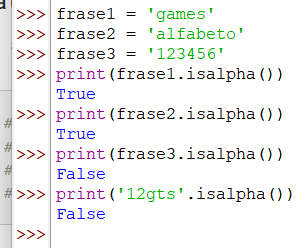
## Métodos upper() e lower():

O método upper() recebe o valor de uma string e retorna ela mesma, mas com todos os caracteres em maiúscula.

E o método lower é justamente o contrário, ele recebe o valor da string e retorna ela mesma com todos os caracteres em minúsculas.

## Método isalpha():

O método isalpha retorna “True” se todos os caracteres na string forem alfabetos, caso contrário, retorna “False”. Esta função é usada para verificar se o argumento inclui apenas caracteres do alfabeto.

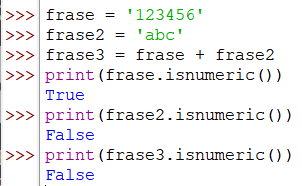


Esse método é bastante utilizado para saber se o que foi digitado (em string/str) foi um número ou não, e se retorna False, ele converte para int.

## Método isnumeric():

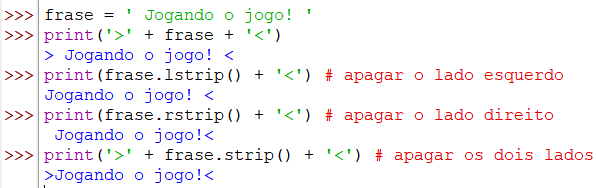
Isnumeric é um método embutido usado para a manipulação de strings, onde o método retorna “True” se todos os caracteres da string forem numéricos, caso contrário, retorna “False”.

Essa função é usada para verificar se o argumento contém todos os caracteres numéricos, como: inteiros, fações, subscrito, sobrescrito, numerais romanos etc.

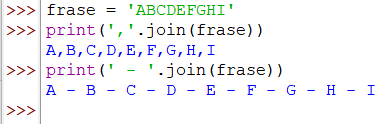
****

## Método strip():

Python é capaz de encontrar espaços em branco dos lados direito e esquerdo de uma string. Strip pega uma cópia da string que tiver esses espaços em branco do início e no final e retorna sem esses espaços.  
Também é possível apagar apenas um dos lados (esquerdo ou direito) utilizando o lstrip() - lado esquerdo – e o rstrip() - lado direito



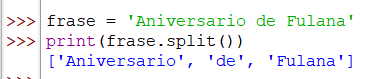
## Método join():

O método join é usado para especificar os elementos de uma sequência de caracteres para gerar uma nova sequência de conexão. Ou seja, junta cada item da string com um delimitador especificado. É o inverso do split() e também é aceito em listas.

## Métodos Split

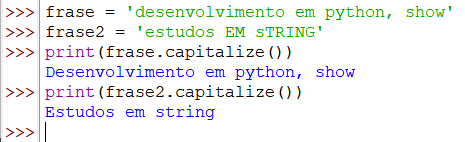
O método split é uma das funções disponíveis em Python utilizada para a manipulação de strings. Na prática, ele permite dividir o conteúdo da variável de acordo com as condições especificadas em cada parâmetro da função ou com os valores predefinidos por padrão.

O método split() retorna uma lista de strings após quebrar a string dada pelo separador especificado.

****

## Método capitalize():

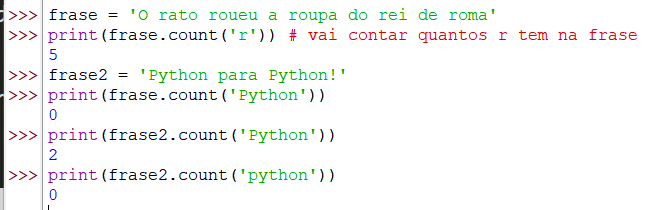
O método retorna uma cópia da string original e converte o primeiro caractere da string em uma letra maiúscula enquanto transforma todos os outros caracteres na string em letras minúsculas.

****

## Método count():

Count é uma função embutida na linguagem de programação Python que retorna o número de ocorrências de uma substring na string fornecida. O método count() retorna um inteiro que denota o número de vezes que uma substring ocorre em uma determinada string.

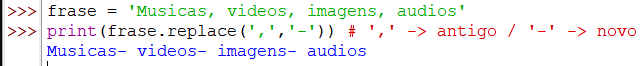
Também a diferenciação de letras maiúscula e minúsculas.

****

## Método replace():

O método replace(), recebe dois argumentos, para fazer a localização e substituir valores dentro de uma string.

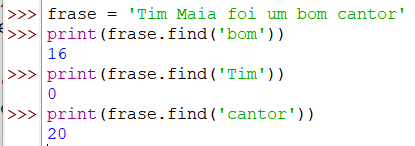
Na sua sintaxe, primeiro vem o argumento antigo e logo depois o novo.

****

## Método find():

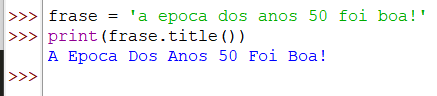
O método find deve ser utilizado apenas se te interessar a posição da ocorrência na string, isto é, saber em que parte da string foi encontrado o valor desejado. Se a intenção é apenas verificar se for encontrado, o ideal é utilizar o operador in.

Usando o find ele vai retornar o menor índice da primeira aparição dessa substring. Se não encontrar retorna -1.

****

## Método title():

A função title é usada para converter o primeiro caractere em cada palavra em maiúsculas e os caracteres restantes em minúsculas na string e retorna uma nova string.

****

# Manipulação de Strings

## Caracteres de escape

Um *caractere de escape* permite usar caracteres que, de outra maneira, não poderiam ser incluídos em uma string. **Um caractere de escape é constituído de uma barra invertida (\) seguida do caractere que você deseja incluir na string.**

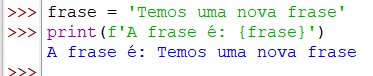
|  |  |
| --- | --- |
| **Caractere de escape** | **Exibido como** |
| \’ | Aspas simples |
| \” | Aspas duplas |
| \t | Tabulação |
| \n | Quebra ou mudança de linha |
| \\ | Barra Invertida |

## Strings de múltiplas linhas com aspas triplas

# F-String:

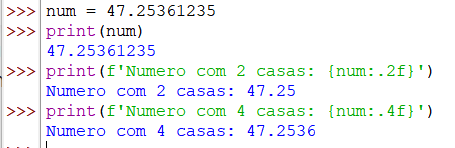
A partir das versões mais recentes do Python, começou-se a usar uma ferramenta para formatação de textos, só que ela não serve apenas e exclusivamente para formatação de textos, ela da mais liberdade para se modificar uma string.

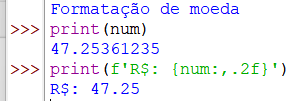
## F-string e suas formatações:

As f-strings vão servir para que você consiga colocar uma variável dentro de um texto, e isso é feito utilizando a letra “f” antes do texto e colocando a sua variável dentro de {} (chaves).

Com isso também nós facilita muito a formatação de outras informações, deixando bem mais fácil nossa vida de programador.

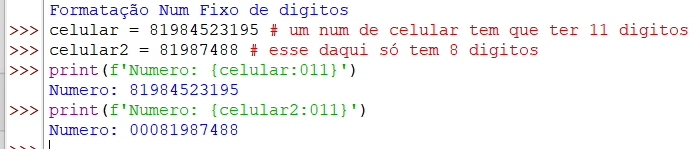
**Exemplo de formatação com casas decimais:**

****

****

**Exemplo de formatação de Moeda:**

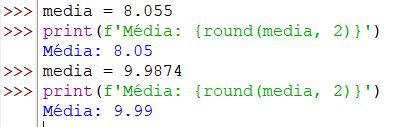
**Exemplo de Número fixo de dígitos:**

****

## Método round() no f-string

Você pode usar a função round() do Python para arredondamento, escolhendo o número de casas que desejar.

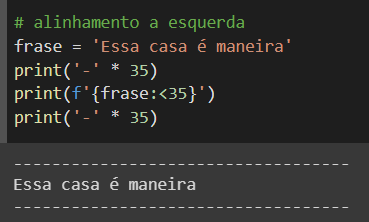
A função round() recebe 2 argumentos. O primeiro é o número que deseja arredondar, e o segundo é a quantidade de casas decimais que você deseja.

****

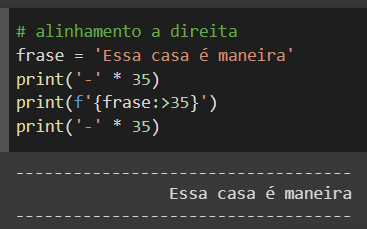
## Alinhamento a Esquerda, Direita e Centro:

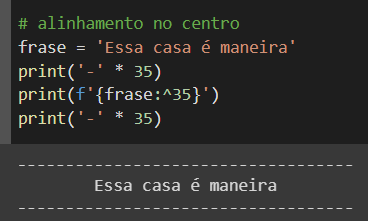
Alinhamento de texto em Python é útil para imprimir saídas formatas e limpas. Algumas vezes, os dados a serem impressos variam em comprimento, o que os torna confusos quando impressos.

Usando o alinhamento de f-string, a saída pode ser alinhada definindo o alinhamento como esquerdo, direito ou centro e também definindo o espaço (largura) a ser reservado para a string.



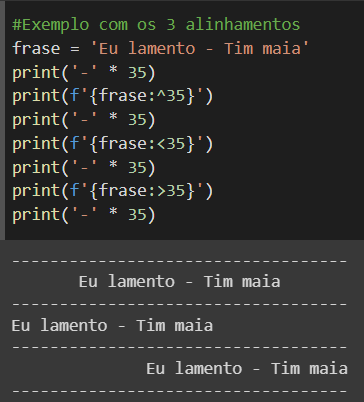
Para a sintaxe da string de saída do alinhamento à **esquerda**, defina ‘<’ seguido pelo número da largura.

Para a sintaxe da string de saída de alinhamento à **direita**, defina ‘>’ seguido pelo número da largura.

****

Para a sintaxe da string de saída do alinhamento **central**, defina ‘^’ seguido pelo número da largura.

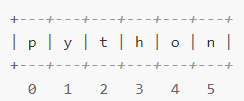
**Exemplo com todos os 3:**

****

# Fatiamento de Strings

O fatiamento consiste em obter uma sub-string de uma determinada string, dividindo-a respectivamente do início ao fim. Selecionar uma fatia é como selecionar um caractere.

Para fatiar uma string, precisamos entender que em toda string a uma sequência. Dentro de uma string a índices de determinada sequência a partir do zero.

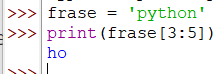
Ilustração 1: Exemplo de sequência com a string "python".

Fatiamento de string é muito utilizado em tuplas, listas e dicionários.

Outro coisa que precisamos saber sobre fatiamento de sequências é:

* Ao fatiar uma string teremos como resultado uma nova string fatiada;
* Ao fatiar uma lista teremos como resultado uma nova lista com os “pedaços”.

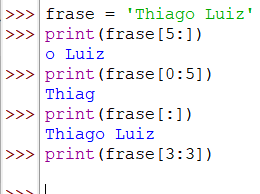
## Como fatiar em python

Dentro de python, a utilização dos índices que vimos anteriormente vai nos auxiliar para conseguir ‘capturar’ a sub-string que queremos.

Vamos utilizar o exemplo da ilustração 1, eu quero pegar apenas as letras do índice 3 até o 5.

Lembrando que, em python ele vai pegar até um número antes. Então ele pegou apenas as letras ‘h’ e ‘o’.

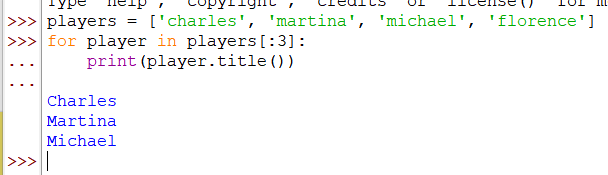
O operador **[n:m]** retorna a parte da string do “enésimo” caractere ao “emésimo” caractere, incluindo o primeiro, mas excluindo o último. Este comportamento é contraintuitivo, porém pode ajudar a imaginar os índices que indicam a parte entre os caracteres.



**Obs.:** Se o primeiro índice for maior ou igual ao segundo, o resultado é uma string vazia, representada por duas aspas

## Percorrer uma fatia com um laço

Você pode usar uma fatia em uma laço for se quiser percorrer um subconjunto de elementos de uma lista.

****

As fatias são muito úteis em várias situações. Por exemplo, ao trabalhar com dados, você pode usar fatias para processar seus dados em porções de tamanho específico.

Ou então, quando criar um jogo, você poderia adicionar a pontuação final de um jogador em uma lista sempre que esse jogador acabar de jogar. Seria possível então obter as três pontuações mais altas de um jogador ordenando a lista em ordem decrescente e obtendo uma lista que inclua apenas as três primeiras pontuações.

# Utilizações em Listas

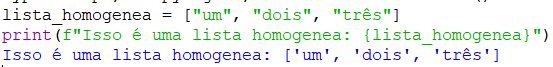
Dentro da linguagem Python, o tipo de dado nativo para agrupar valores, que é mais versátil é a **lista**. Ela serve para armazenar informações complexas de forma organizada.

As listas em Python ajudam a estruturar e classificar o conteúdo para uso posterior. Ao contrário de *tuple* (tupla), *set e dicionário*, uma lista pode armazenar dados de vários tipos, “aninhando diferentes valores e listas.

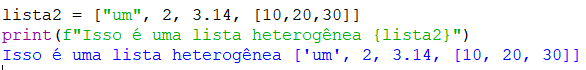
Uma lista é uma estrutura de dados muito flexível. Você pode editá-las, organizá-las, ampliá-las ou encurtá-las através de vários métodos.

Uma lista é uma sequência de valores. É um tipo de “contêiner” usado para armazenar um número indefinido de dados arbitrários do mesmo tipo ou de tipos diferentes.

**Exemplo de lista homogênea** – *que é uma lista que contém elementos do mesmo tipo*:



**Exemplo de lista heterogênea** – *combina diferentes tipos de elementos, e nesse exemplo teremos uma string, um número inteiro, um ponto flutuante e outra lista:*

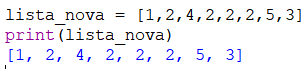


As listas em Python tem as seguintes propriedades:

* Elas são **ordenadas**: *cada elemento dentro da lista está indexado e mantém uma ordem definida;*
* Permitem elementos **duplicados**: *as listas podem ter elementos com o mesmo valor;*
* São **mutáveis**: *podemos editar, excluir ou adicionar elementos a uma lista existente.*

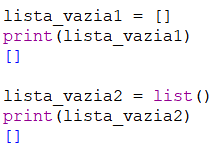
## Criando uma lista

Para se criar uma lista em Python, use colchetes **[ ]** que incluam itens separados por vírgulas:



Outra forma de criar uma lista é usando o construtor de classe *list* (iterable), que converte um contêiner iterável em uma lista:

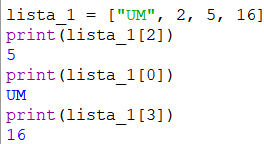
Existe também a **lista vazia**. Ela é uma lista que não contém itens, e é usada para adicionar dados posteriormente. Você pode criá-la usando colchetes vazios e o construtor *list()*:

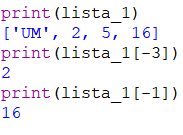


## Acessando elementos em uma lista

Para acessar um elemento de uma lista, são aplicados índices. Eles são números inteiros que indicam a posição de um elemento em uma lista.

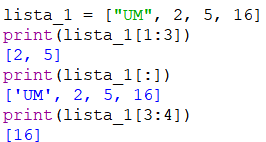
Para selecionar um elemento, especifique seu índice entre colchetes. A indexação varia de 0 a n-1, onde n é o tamanho da lista.

Por exemplo, em uma lista com 3 elementos, os respectivos índices de cada um dos itens seriam 0, 1 e 2.



A uma observação, a qual é que o Python também permite usar índices negativos para acessar os elementos de uma sequência. Neste caso, o índice [-1] se refere ao último elemento da lista, o [-2] ao penúltimo, etc.

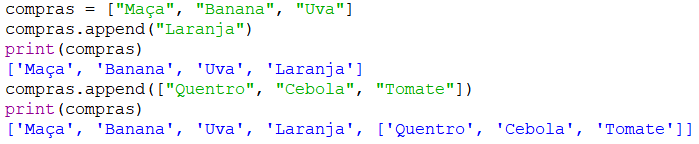
Além disso, você pode fatiar uma lista pelo intervalo de elementos ou criar sublistas menores de uma lista usando o operador **[:]**, indicando à esquerda o valor inicial e, à direita, o valor final que não está incluído. Assim, [0:2] você criará uma lista com os elementos [0] e [1] da lista original.



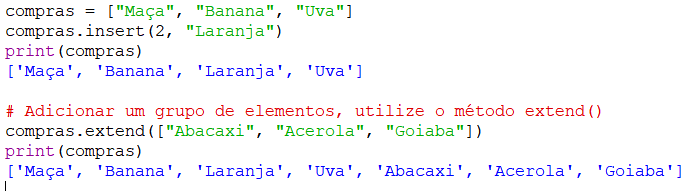
## Adicionando elementos na lista (*append e insert*):

As listas em Python são dinâmicas, permitindo que você adicione e remova elementos. Há várias formas de adicionar novos dados a listas existentes.

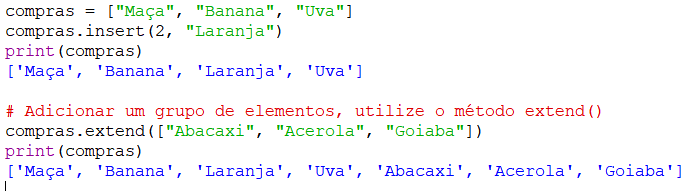
Para adicionar um novo elemento a uma lista, aplique o método ***append()***. Adicione o novo elemento ao final da lista sem afetar nenhum dos elementos anteriores. Este método é preferível quando você deseja adicionar elementos a uma lista vazia, quando necessário.



Também é possível adicionar um elemento em uma posição específica com o método ***insert(índice, elemento)*** onde você especifica a posição do elemento e o novo valor. Os elementos com o índice maior que o indicado se movem para a direita.



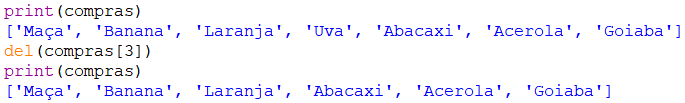
## Juntando duas listas (*extend*):



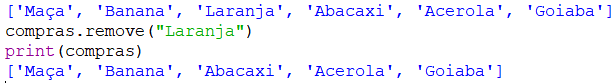
## Excluindo elementos de uma lista:

Em Python existem quatro maneiras principais de excluir itens de uma lista:

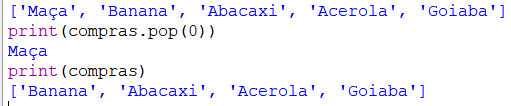
* Por seu índice usando a função **del()**:



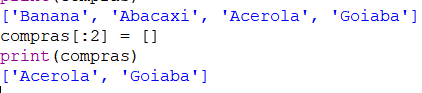
* Por seu valor usando o método **remove()**:



* Usando o método **pop()**, que além de excluir, recupera o elemento realizando uma extração:

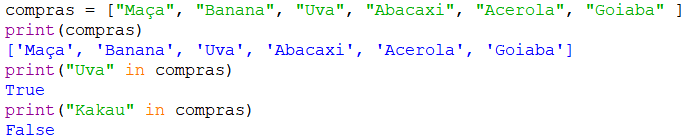


* Por sua classificação mediante fatiamento de listas:



## Como saber se um elemento está na lista?

Para comprovar se um item específico está na lista, aplique o operador de associação “in”. Se um elemento está armazenado na lista, a expressão booleana ***True*** será retornada, caso contrário, será retornado ***False.***



## Método Sort()

Esse método é usado para ordenar uma lista diretamente, o que significa que ela realiza a mutação da lista ou que a modifica diretamente sem criar cópias adicionais. Ela pode ordenar uma lista em ordem ascendente ou ordem descendente.

Quando o método sort() é utilizado ele modifica a lista e, portanto, sua versão original se perde. Por causa disso, você somente deve usar esse método se:

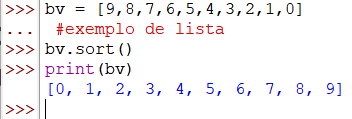
* Quiser modificar (ordenar) a lista permanentemente;
* Não precisar manter a versão original da lista.

A sintaxe do método é bem simples: *<list>.sort()*

É possível passar argumentos, mas caso você não passe, por padrão:

* A lista será ordenada em ordem ascendente;
* Os elementos da lista serão comparados diretamente usando seus valores com o operador <.

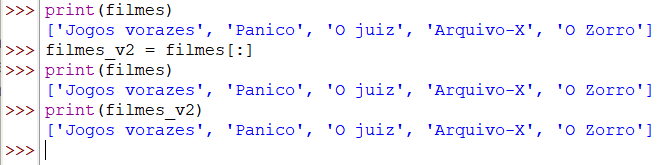
**Exemplo:**



## Copiando uma lista

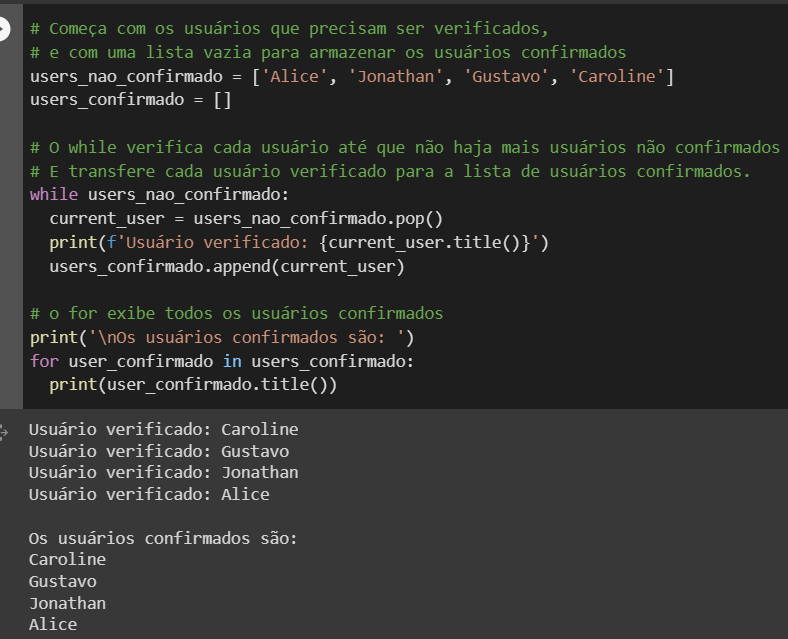
Com frequência, você vai querer começar com uma lista existente e criar uma lista totalmente nova com base na primeira.

Para criar uma lista, podemos criar uma fatia que inclua a lista original inteira omitindo o primeiro e o segundo índices ([:]). Isso diz a Python para criar uma lista que começa no primeiro item e termina no último, gerando uma cópia da lista toda.



## Transferindo itens de uma lista para outra:

Considere uma lista de usuários recém-registrados em um site, porém não verificados. Depois de conferir esses usuários, como podemos transferi-los para uma lista separada de usuários confirmados? Uma maneira seria usar um laço while para extrair os usuários da lista de usuários não confirmados à medida que os verificarmos e então adicioná-los em uma lista separada de usuários confirmados.

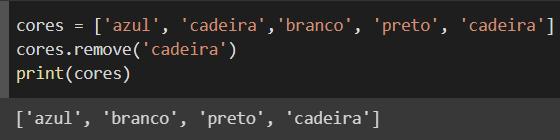


O laço while vai executar o looping enquanto a lista *users\_nao\_confirmado* não estiver vazia. Nesse laço, a função pop() remove os usuários não verificados, um de cada vez, do final de *users\_nao\_confirmado*. Nesse caso, como Caroline é o último elemento da lista *users\_nao\_confirmado*, sei nome será o primeiro a ser removido, armazenando em *current\_user* e adicionando à lista *users\_confirmado*. Assim segue-se com os outros elementos dentro da lista.

Quando a lista de usuários não confirmados estiver vazia, o laço para e a lista de usuários confirmados é exibida.

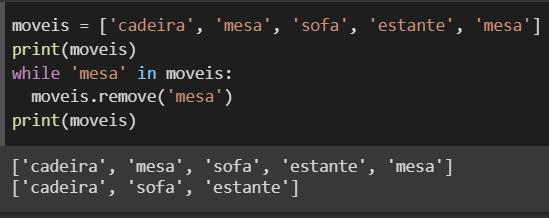
## Removendo instâncias de uma lista:

### Com *remove()*

Assim como é possível adicionar instâncias dentro de uma lista, também é capaz de removê-las. O método remove() é bem simples de se usar, como veremos.

Contudo, caso haja mais de uma instância com o mesmo nome, o comando remove apenas apagará uma única vez. Como podemos ver no exemplo, dentro da lista *cores* havia duas instâncias com o nome cadeira, uma no índice 1 e a outra no índice 4. Mas, quando foi utilizado o método remove(), apagou-se apenas o que apareceu primeiro.

Para apagar todas as instâncias que forem repetidas, basta colocar um laço while até que a instância não esteja mais na lista.

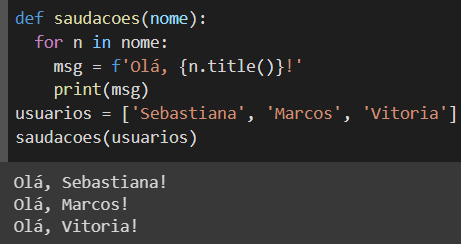


### Com pop():

## Passando uma lista para uma função

Com frequência, você achará útil passar uma lista para uma função, seja uma lista de nomes, de números ou de objetos mais complexos, como dicionários. Se passarmos uma lista a uma função, ela terá acesso direto ao conteúdo dessa lista.

Suponha que tenhamos uma lista de usuários e queremos exibir uma saudação a cada um. O exemplo a seguir envia uma lista de nomes a uma função chamada saudacoes(), que saúda cada pessoa da lista individualmente.



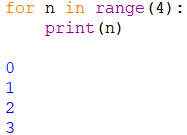
## Descompactação de Zip (listas)

## Trabalhando com Listas

### Utilizando loops for com listas

O uso de loops for tem a missão para executar um bloco de código um número específico de vezes. Tecnicamente, um loop for repete o bloco de código uma vez para cada valor de uma lista ou de um valor semelhante a uma lista.

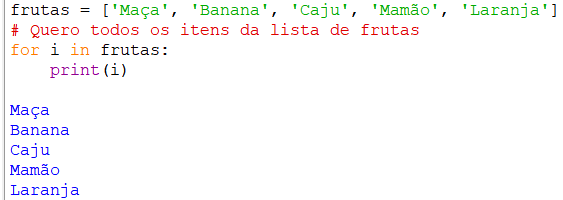
Por exemplo:



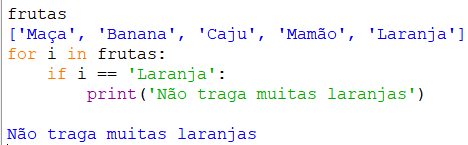
|  |
| --- |
| O retorno de range(4) é um valor semelhante a uma lista <[0, 1, 2, 3]>. |

* Uma técnica comumente utilizada é utilizar, no lugar do range, a lista ao qual quer percorrer.

EX:



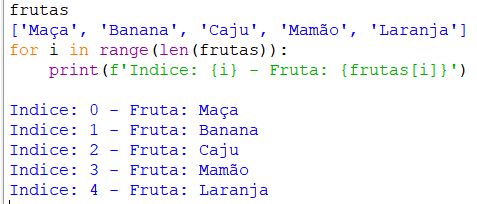
* Veja algo mais interessante, utilizando condicionais:



Aqui ele percorreu toda a lista, e quando encontrou a palavra ‘Laranja’, que foi o que a condicional exigiu, ele apresentou o if na tela.

* Outra técnica comumente utilizada consistem em usar <*range(len(list))*> com um loop for para fazer uma iteração pelos índices de uma lista.

EX:



### Operadores IN e NOT IN

# Utilizações em Tuplas

# Utilizações em Dicionários

Dicionários são capazes de armazenar uma quantidade quase ilimitada de informações, mostraremos como percorrer os dados de um dicionário com um laço. Entender os dicionários permite modelar uma diversidade de objetos do mundo real de modo mais preciso.

Um dicionário em Python é uma coleção de pares chave-valor – denotada na literatura pelos termos em inglês *key:value*. Cada chave é conectada a um valor, e você pode usar uma chave para acessar o valor associado a ela.

O valor de uma chave pode ser um número, uma string, uma lista ou até mesmo outro dicionário.

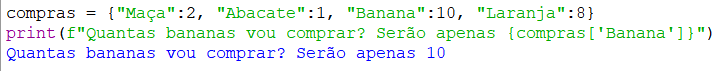
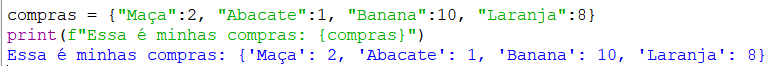
Um dicionário é representado entre **chaves {}**, com uma série de pares chave-valor entre elas.

Vale comentar também algumas vantagens em se utilizar dicionários:

* **São mutáveis**, pois permitem remover ou modificar os elementos do dicionário a qualquer momento;
* **São dinâmicos**, já que não exige especificar o tamanho ou o tipo dos elementos do dicionário antecipadamente;
* **São eficientes na hora de pesquisar**, **inserir ou deletar** elementos do dicionário em tempo constante, independentemente do tamanho do dicionário;
* **São versáteis**, possibilitando usar qualquer tipo de objeto imutável como chave e qualquer tipo de objeto como valor. Também permitem aninhar dicionários dentro de outros dicionários para criar estruturas de dados complexas.

## Criando um dicionário

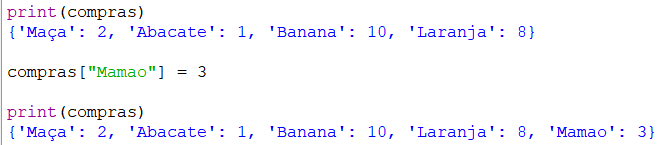
A criação de um dicionário é simples, ele segue a sintaxe: **nome\_dicionario = {*key:valor*}.** Veja o exemplo abaixo:



## Adicionando elementos em um dicionário já criado

Para incluir novos valores em um dicionário, basta atribuir um valor a uma nova chave ou a uma chave existente. A sintaxe básica para adicionar um novo par chave-valor a um dicionário é:

Exemplo:

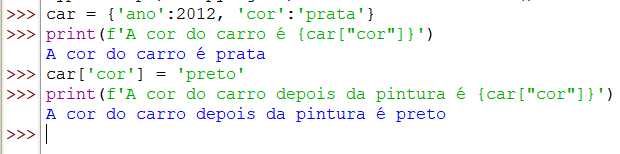


<http://www.bosontreinamentos.com.br/programacao-em-python/16-python-dicionarios-parte-2-aninhamento-de-dicionarios-e-listas-inclusao-de-valores/>

## Modificando valores em um dicionário

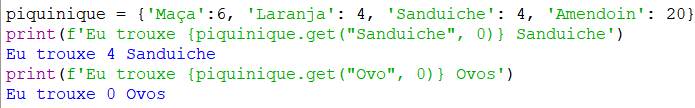
Para modificar um valor em um dicionário, especifique o nome do dicionário com a chave entre colchetes e o novo valor que você quer associar a essa chave.

Como um exemplo, imagine um carro da cor prata, que depois da pintura ficou com a cor preta;

****

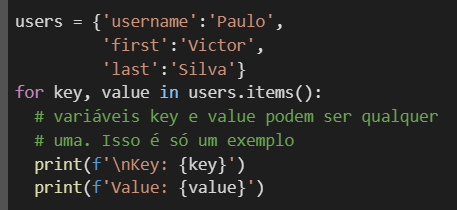
## Verificando se um valor e chave estão no dicionário (get)

É tedioso verificar se uma chave está presente em um dicionário antes de acessar o valor dessa chave. Felizmente, os dicionários têm um **método get()** que aceita dois argumentos: *a chave do valor a ser obtido e um valor alternativo a ser retornado se essa chave não existir*.



## Percorrendo um dicionário com um laço

Um único dicionário python pode conter apenas alguns pares chave-valor ou milhões deles. Como um dicionário pode conter uma grande quantidade de dados, python permite percorrer um dicionário com um laço.   
Dicionários podem ser usados para armazenar informações de várias maneiras; assim, há diversos modos diferentes de percorrê-los com um laço.

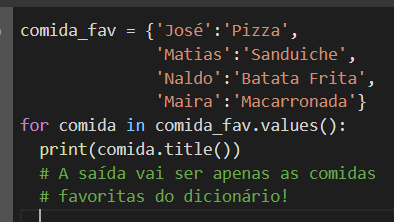
****

### Percorrendo todas as chaves com um laço

O método keys() é conveniente quando não precisamos trabalhar com todos os valores de um dicionário.

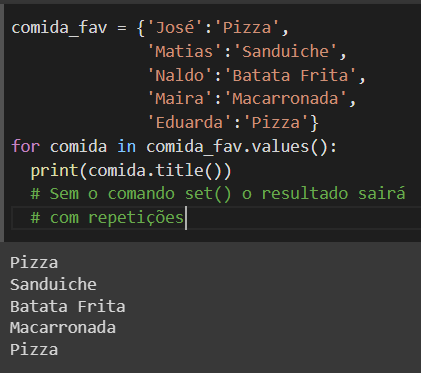
O método keys() não serve apenas para laços: na verdade, ele devolve uma lista de todas as chaves.

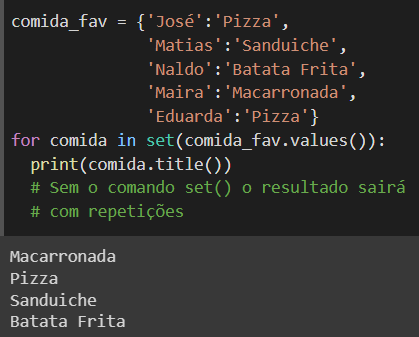
### Percorrendo todos os valores com um laço com values() e set()

Se você estiver ais interessando nos valores contidos em um dicionário, o método values() pode ser usado para devolver uma lista de valores sem as chaves.

Essa abordagem com **o método values()** extrai todos os valores do dicionário, sem verificar se há repetições. Isso pode funcionar bem com uma quantidade pequena de valores, mas imagine o exemplo anterior, contendo repetições de comidas favoritas, o resultado seria uma lista com muitas repetições.

Para ver cada valor sem repetições, podemos usar um conjunto (set). Um conjunto é semelhante a uma lista, exceto que cada item de um conjunto deve ser único.

Percorrendo valores sem o método **set()**

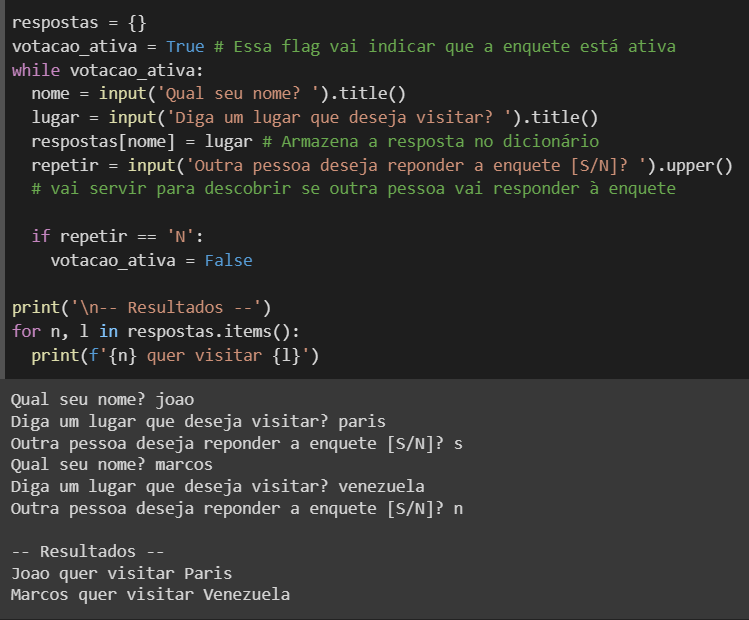
Percorrendo valores **com** o método **set()**

Quando colocamos set() em torno de uma lista que contenha itens duplicados, Python identifica os itens únicos na lista e cria um conjunto a partir desses itens.

## Preenchendo um dicionário com input

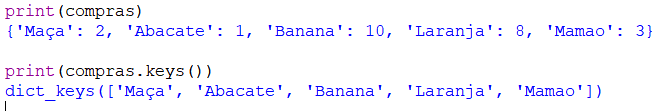
Como já é ciente, também é possível criar-se um dicionário em que nele, as chaves e valores sejam o próprio usuário que digite.

Vamos criar um exemplo em que haja uma enquete em que solicita ao usuário o nome e uma resposta. Armazenaremos os dados coletados em um dicionário, pois queremos associar cada resposta a um usuário em particular.

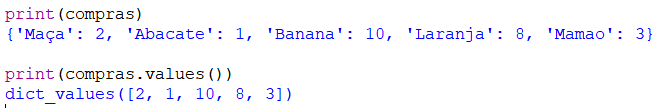


## Métodos úteis nos dicionários

* **Keys**
  + keys() retorna uma lista com todas as chaves do dicionário.



* **Values**
  + values() retorna uma lista com todos os valores do dicionário



* **items**
* **fromkeys**
* **copy**
* **clear**
* **get**
* **update**
* **pop**

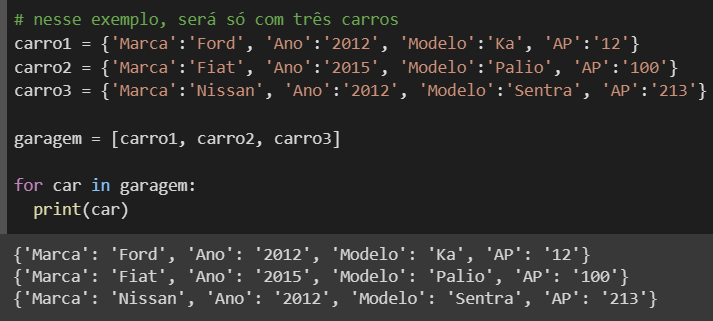
# Informações Aninhadas

Às vezes, você vai querer armazenar um conjunto de dicionários em uma lista ou uma lista de itens como um valor em um dicionário. Isso é conhecido como aninhar informações. Podemos aninhar um conjunto de dicionários em uma lista, uma lista de itens em um dicionário ou até mesmo um dicionário em outro dicionário. Aninhar informações é um recurso eficaz.

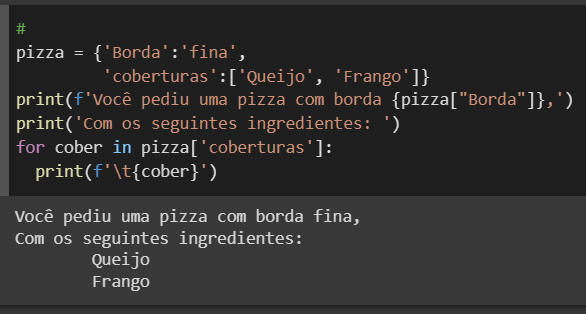
## Uma lista de dicionários

É comum armazenar vários dicionários em uma lista quando cada dicionário tiver diversos tipos de informação sobre um objeto. Todos os dicionários da lista devem ter uma estrutura idêntica para que possamos percorrer a lista com um laço e trabalhar com cada objeto representado por um dicionário do mesmo modo.

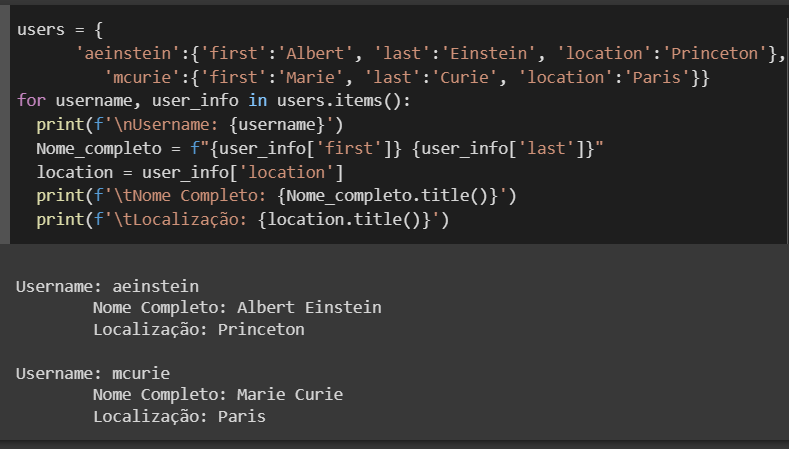
Vamos imaginar uma garagem de um prédio com mais de 10 carros. Um dicionário contém várias informações sobre um só carro, mas não haveria como adicionar mais informações sobre um segundo modelo de carro.   
Uma maneira de contornar isso, é criar uma lista de carros, em que cada carro seja representado por um dicionário com informações sobre ele.



## Uma lista em um dicionário

Em vez de colocar um dicionário em uma lista, às vezes é conveniente colocar uma lista em um dicionário. Vamos usar um exemplo de que alguém está pedindo uma pizza. Se usássemos apenas uma lista, tudo que poderíamos realmente armazenar é uma lista dos ingredientes da pizza. Com um dicionário, uma lista de ingredientes pode ser apenas um dos aspectos da pizza que estamos descrevendo.

## Um dicionário em um dicionário

Podemos aninhar um dicionário em outro dicionário, mas o código poderá ficar complicado rapidamente se isso for feito.  
Por exemplo, se você tiver vários usuários em um site, cada um com um nome único, os nomes dos usuários poderão ser usados como chaves em um dicionário. Você poderá então armazenar informações sobre cada usuário usando um dicionário como o valor associado a cada nome de usuário.

# Módulos em Python

Importar um módulo em seu programa principal é dizer ao Python para deixar o código de seu módulo disponível no arquivo de programa em execução no momento. Alguns desses módulos (ou bibliotecas) são disponíveis por padrão quando instalamos o Pyhton, outros temos que instalar manualmente.

Ao decorrer dessa seção eu anotei alguns módulos vistos em aula no Curso de Python do Luiz Otávio, na Udemy.

**Assistir aulas 154 e 155 para entender melhor também sobre importação de funções próprias.**

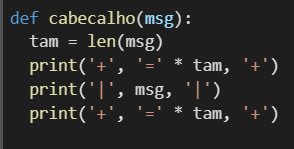
## Importando uma função (própria) para um módulo

Uma vantagem das funções é a maneira como elas separam blocos de código de seu programa principal. Ao usar nomes descritivos para suas funções, será bem mais fácil entender o seu programa principal. Você pode dar um passo além armazenando suas funções em um arquivo separado chamado módulo e, então, importar esse módulo em seu programa principal.

Armazenar suas funções em um arquivo separado permite ocultar os detalhes do código de seu programa e se concentrar na lógica de nível mais alto. Também é permitido reutilizar funções em muitos programas diferentes.

Quando armazenamos funções em arquivos separados, podemos compartilhar esses arquivos com outros programadores sem a necessidade de compartilhar o programa todo.

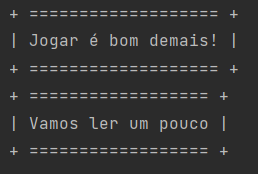
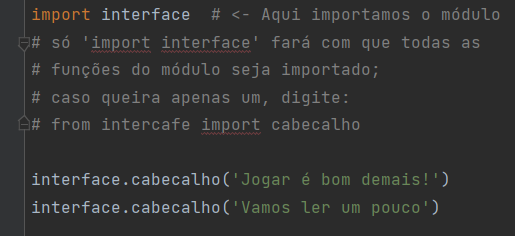
Para começar a importar funções, inicialmente precisamos criar um módulo. Um módulo é um arquivo terminado em *.py* que contém o código que queremos importar para o nosso programa.



Para nosso exemplo, criaremos um módulo chamado *interface.py*. Que nele conterá:

Agora criaremos um arquivo separado, eu chamarei de *programa\_principal.py* no mesmo diretório que este *interface.py*.

Nesse arquivo, importaremos o módulo que acabamos de criar e, em seguida, faremos duas chamadas para apresentar um cabeçalho para nossa mensagem.

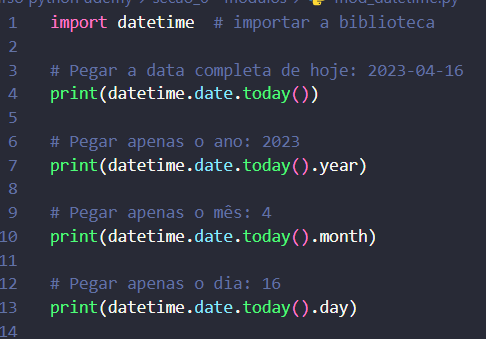


## Módulo datetime (*para datas e horas*)

O módulo datetime é da biblioteca nativa do Python e serve para trabalhar com datas e horas.

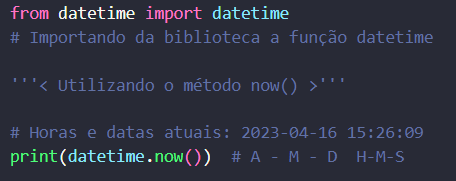
Documentação do datetime: [https://docs.python.org/pt-br/3/library/datetime.html#module-datetime](https://docs.python.org/pt-br/3/library/datetime.html" \l "module-datetime)

Vamos utilizar o exemplo no qual precisamos pegar a data do dia atual, para isso vamos primariamente importar a biblioteca e logo depois chamar as funções do módulo. Vejamos o exemplo:



Dentro do módulo *datetime* também tem a possibilidade de utilizarmos e trabalharmos com horários (e fuso horários ou *timezones*). O módulo nos dá uma solução muito mais apropriada para o nosso problema com o tipo *datetime* – sim, tem o mesmo nome do módulo, cuidado com a confusão!

Uma das vantagens da classe *datetime* é que ela consegue cuidar da data e do horário ao mesmo tempo. A única diferença em nosso uso é que, em vez do método *today()*, usaremos o método *now()*:



### Problema do fuso horário

Ver aula 275 para falar sobre isso

### Calculando datas

## Módulo Calendar (*trabalhar com calendários*)

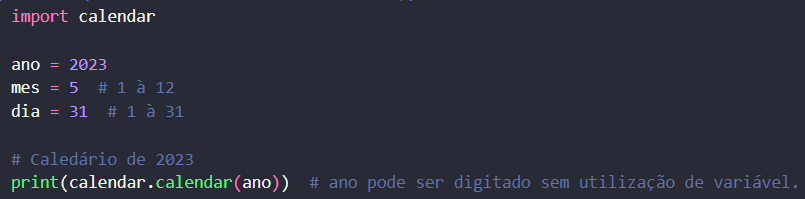
O python possui um módulo interno para que possamos trabalhar com operações de calendário, o seu nome é *calendar*. Dentro dele podemos utilizar várias coisas dedicadas a calendários e datas.

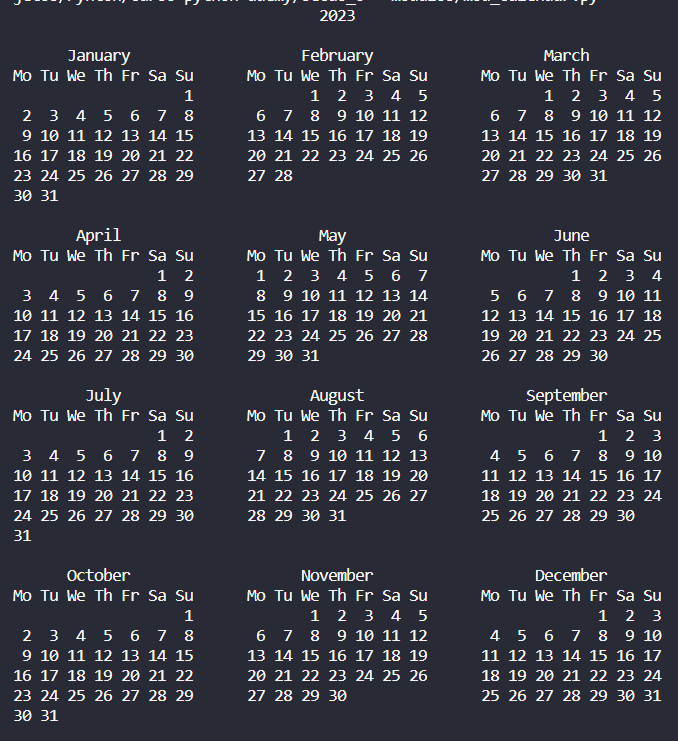
OBS: Por padrão a semana começa na Segunda-Feira e o o último dia é o Domingo;

Documentação de *calendar*: [https://docs.python.org/pt-br/3.7/library/calendar.html#module-calendar](https://docs.python.org/pt-br/3.7/library/calendar.html" \l "module-calendar)

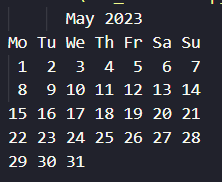
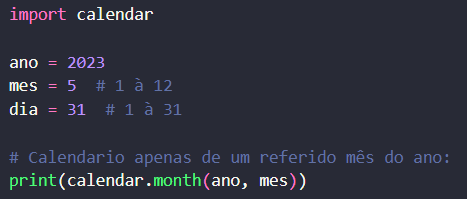
Com o calendar, você pode saber coisas como:

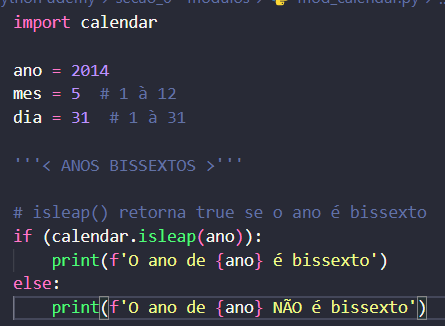
1. Qual o último dia do mês (monthrange)
2. Qual o nome e número do dia de determinada data (weekday)
3. Criar um calendário em si (monthcalendar)
4. Trabalhar com coisas específicas de calendários (calendar, month)



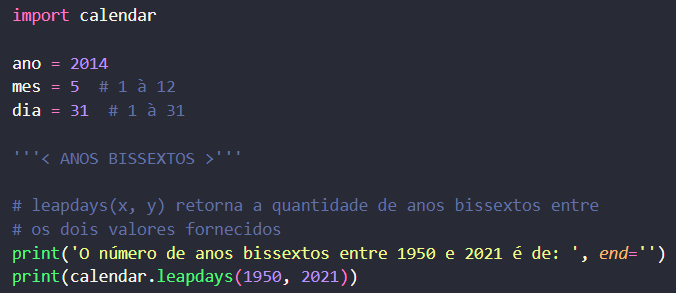


Resultado:



**Anos Bissextos:**

****

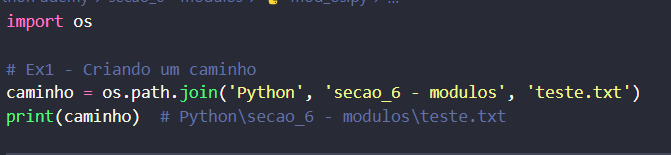


## Módulo os

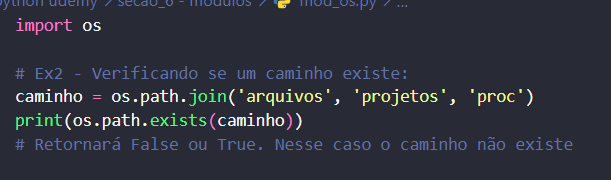
O módulo **os** é um dos principais em Python, ele traz informações sobre o sistema operacional e traz funcionalidades para mexer com o sistema operacional.

Documentação: [https://docs.python.org/pt-br/3/library/os.html#module-os](https://docs.python.org/pt-br/3/library/os.html" \l "module-os)

Dentro do módulo os existe outro módulo, o **os.path** que é muito útil para se trabalhar com caminhos de arquivos. Isso é útil pois para o Windows, Max ou Linux tendem a ter diferenças com os caminhos dos arquivos. Vamos ver um exemplo:



Verificando se um caminho existe:



Também é possível navegar entre caminhos e pastas: (*aula 283 e 284*)

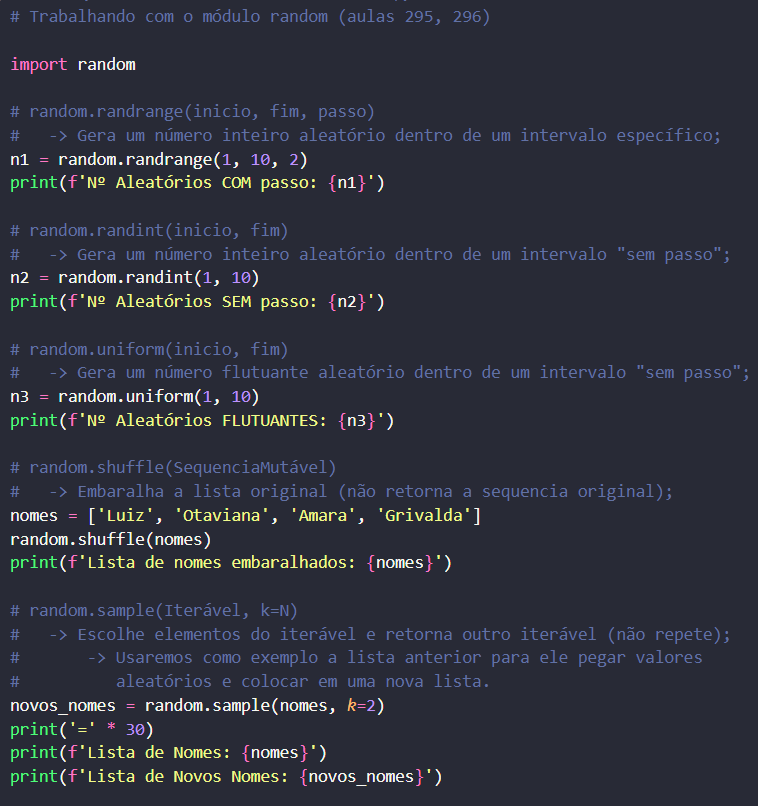
## Módulo Random

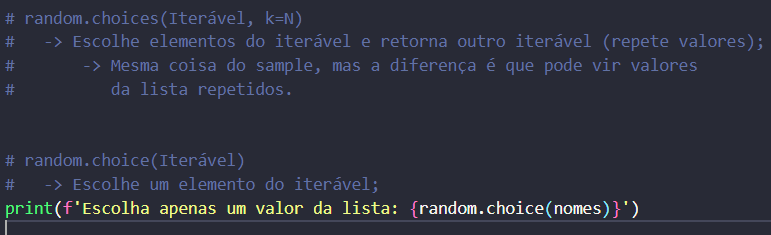
Documentação: [https://docs.python.org/pt-br/3.7/library/random.html#module-random](https://docs.python.org/pt-br/3.7/library/random.html" \l "module-random)

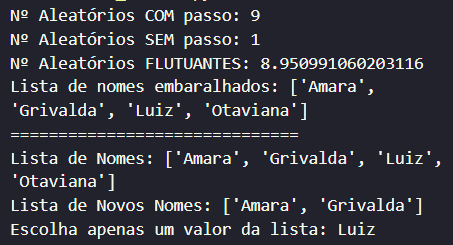
Random é um módulo que é utilizado para gerar números pseudoaleatórios. Também podemos selecionar os elementos de uma lista de forma aleatória ou exibir o seu resultado embaralhado.

Obs.: números pseudoaleatórios significam que os números parecem ser aleatórios, mas na verdade não são. Portanto, este módulo não deve ser usado para segurança ou uso criptográfico. O motivo disso é que quando temos uma mesma entrada e um mesmo algoritmo, a saída pode ser previsível.

Vamos ver agora as funções disponíveis dentro do módulo range e o que dá para fazer:







## Módulo secrets

## Módulo subprocess

Dentro do Python existe um módulo responsável por executar programas externos e ler suas saídas em seu script python. Como já disse, com o subprocess podemos ser capazes que criar processos, executando programas externos ao nosso código e permitindo que analisemos a saída que tal programa geraria na tela.

O subprocess inclui várias classes e funções, mas falarei da mais simples e útil dela, que é o subprocess.run().

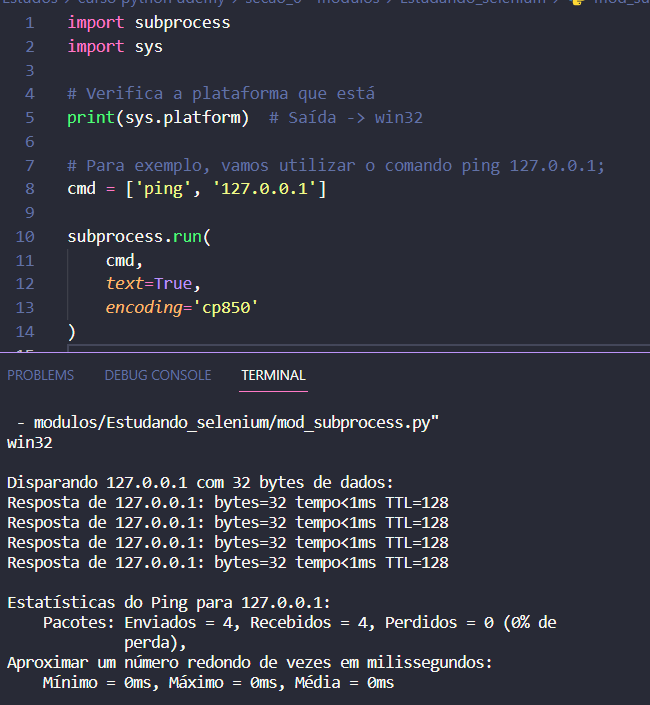
**Alguns argumentos principais de subprocess.run()**:

* stdout, stdin e stderr → Redirecionam saída, entrada e erros;
* capture\_output → Captura a saída e erro para uso posterior;
* text → Se True, entradas e saídas serão tratadas como texto e automaticamente codificadas ou decodificadas com o conjunto de caracteres padrão da plataforma (*geralmente UTF-8*);
* shell → Se True, terá acesso ao shell do sistema;

**OBS:** *nunca passe entrada não confiável para subprocess.run. Como o subprocess.runtem a capacidade de executar comandos arbitrários em seu computador, os indivíduos maliciosos podem usá-lo para manipular seu computador de maneiras inesperadas.*

**OBS\_2:** *Importante relatar que a codificação de caracteres do Windows pode ser diferente (as acentuações) por isso, é opcional, mas, caso seja necessário pode-se passar uma dessas codificações para ver se funciona: cp1252, cp852, cp850 ou outros.*

**Exemplo**:



**+ Informações:**

**blog**: https://www.digitalocean.com/community/tutorials/how-to-use-subprocess-to-run-external-programs-in-python-3-pt

**Doc. do módulo**: <https://docs.python.org/pt-br/dev/library/subprocess.html>

## Módulo PyPDF2

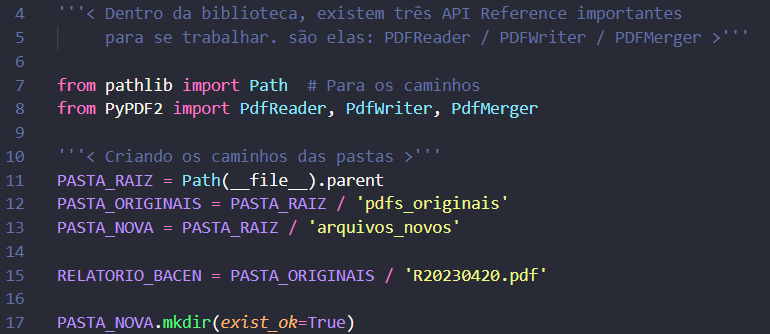
Doc.: <https://pypdf2.readthedocs.io/en/3.0.0/>

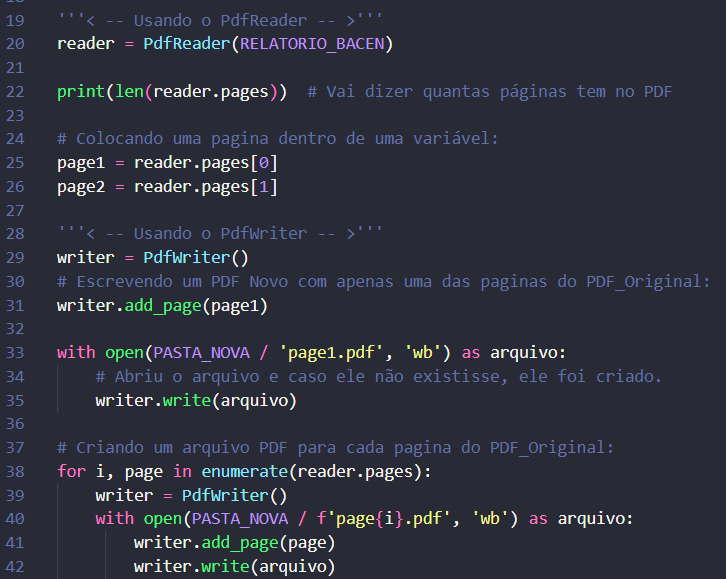
Importante ler a documentação, pois nem todo PDF é igual

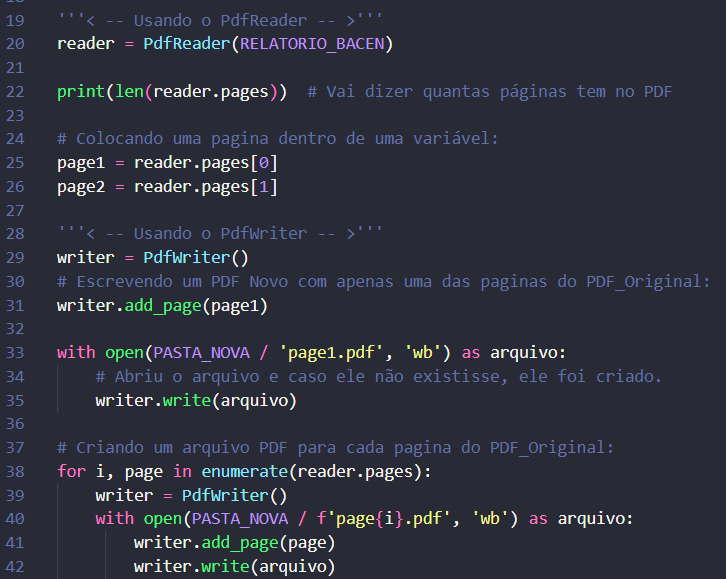
PyPDF2 é uma biblioteca de manipulação de arquivos PDF feita em Python puro, gratuito e de código aberto. Ela é capaz de ler, manipular, escrever e unir dados de arquivos PDF, assim como adicionar anotações, transformar páginas, extrair texto e imagens, manipular metadados, e mais.

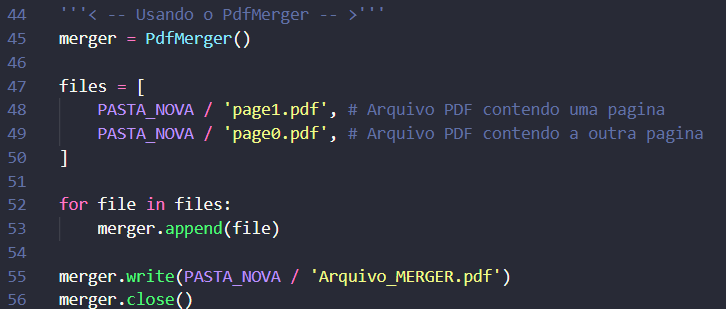
Para instalar o módulo, seja no ambiente virtual ou não, basta digitar: **pip install pypdf2**.

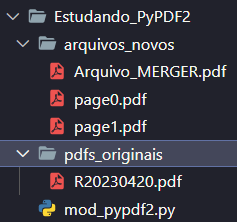
Para nosso exemplo, utilizaremos um PDF do Banco Central, chamado relatório fócus.

Código 1: Parte inicial, onde seleciono as pastas e importo as bibliotecas

Código 2: Utilizando a API PdfReader que server para ler os PDF's

Código 3: Utilizando a API PdfWriter que serve para escrever no PDF

Código 4: Utilizando a API PdfMerger que serve para juntar e fazer outras coisas com o PDF



## Openpyxl – Trabalhando com arquivos Excel, xlsx, xlsm, xltx e xltm.

Aulas 333 à 336 – Curso udemy python

## Pillow – redimensionamento de imagens

Aula 337 – Curso udemy python

# Cores em Python

# Expressões Regulares em Python

# Funções para validação de dados

Enquanto estudava sobre funções em python, me deparei com um exercício no qual o foco era simplesmente a validação de dados. Em python, e com o meu pouco conhecimento que tenho até agora, sei que existem diversas maneiras de se fazer uma validação. Contudo, alguns desses métodos foram maneiras interessantes que eu vim a conhecer, então caso eu encontre outras maneiras de verificar dados, irei complementar nessa seção. Como também complementarei com maneira de como validar outros tipos de dados como CPF, E-mail, Número de telefone etc.

## LeiaInt()

O método leiaInt não está na biblioteca padrão do Python, mas bem que poderia estar, já que facilita e muito nossa vida na hora de validar um dado número para inteiro. Esse método eu vi no exercício 104 do canal do Curso em Vídeo. Todos os créditos vão a ele.

def leiaInt(msg):  
 ok = False # variável puramente para encerrar o laço  
 valor = 0 # variável que armazenará o valor se ele for numerico, no final.  
 while True:  
 n = str(input(msg)) # n vai servir como uma variável qualquer  
 if n.isnumeric(): # Se, a variável n tiver valores numericos, então o programa fará:  
 valor = int(n) # a variável 'valor' vai receber n como inteiro  
 ok = True  
 else:  
 print('Você não digitou um número. Tente Novamente!')  
  
 if ok:  
 break  
 return valor

## valida\_int()

Esse método eu aprendi na disciplina de Lógica de programação e Algoritmos do curso de Análise e desenvolvimento de sistemas, na aula 05, que falava sobre funções. Esse método não é melhor que o anterior (leiaint), mas dependendo do caso, ele pode ser bem proveitoso e, talvez, melhor.

O valida\_int recebe três (3) parâmetros, sendo eles:

1º → um input que recebe a string;

2º → um valor mínimo;

3º → um valor máximo.

def valida\_int(pergunta, min, max):  
 x = int(input(pergunta))  
 while (x < min) or (x > max):  
 x = int(input(pergunta))  
 return x

## Validar CPF

# Orientação a Objetos

Como os estudos de Orientação a Objetos (OO) é um assunto bem extenso e de ‘importância’ para programadores, incluindo principalmente eu, decidi incluir um tópico nesse documento, para anotar meus estudos e também para caso no futuro venha a esquecer sobre o que poderia vir a ser os conceitos iniciais da orientação a objetos. Como meu curso de graduação não adentrou ainda ao assunto nos momentos iniciais desse capítulo, eu estou utilizando diversas fontes da internet, como livros, vídeos e até ajuda dos meus colegas de classe. A você que está lendo (podendo ser eu ou, caso eu coloque no git esse arquivo, outra pessoa) espero que possa ajudar a entender nem que seja o básico sobre OO.

## O que é Programação Orientada a Objetos?

A orientação a objetos é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

Embora a Orientação a Objetos tenha vantagens em relação aos paradigmas que a precederam, existe uma desvantagem inicial: ser um modo mais complexo e difícil de se pensar.

Isso pode ser atribuído à grande quantidade de conceitos que devem ser assimilados para podermos trabalhar orientado a objetos. Isso pode ser atribuído à grande quantidade de conceitos que devem ser assimilados para podermos trabalhar orientado a objetos.

## Classes

Embora POO baseia-se fundamentalmente no termo “Objeto”, tudo começa com a definição do que é uma classe. Antes mesmo de ser possível manipular objetos, é preciso definir uma classe, pois esta é a unidade inicial e mínima de código na OO.

Classe é uma estrutura que abstrai um conjunto de objetos características similares. Uma classe define o comportamento de seus objetos através de métodos e os estados possíveis destes objetos através de atributos. Em outros termos, uma classe descreve os serviços providos por seus objetos e quais informações eles podem armazenar.

O objetivo de uma classe é definir, servir de base, para o que futuramente será o objetivo. É através dela que criamos o “molde” aos quais os objetos deverão seguir.

Uma classe também pode ser definida como uma abstração de uma entidade, seja ela física ou conceitual do mundo real. É através de criação de classes que se conseguirá codificar todas as necessidades de um sistema (software).

Outro fator importante para classes é o seu nome. O nome deve representar bem sua finalidade dentro do contexto ao qual ela foi necessária. Vamos criar um exemplo:

|  |
| --- |
| → Em um sistema de controle de restaurante, temos uma classe chamada *pessoa*, essa classe vai representar todos os indivíduos dali, como clientes, garções, cozinheiros, recepcionistas, entregadores etc. |

## O Atributo

Só definir uma classe não é suficiente para começar a manipulá-las. Devemos também detalhá-las para especificar suas características e o que vai ser colocado ali dentro. Essas características é que vão definir quais informações as classes poderão armazenar e manipular. Na OO estas características e informações são denominadas de atributo.

Atributo é o elemento de uma classe, responsável por definir sua estrutura de dados. O conjunto destes será responsável por representar suas características e farão parte dos objetos criados a partir da classe.

Essa definição deixa bem claro que os atributos devem ser definidos dentro da classe. Devido a isso, são responsáveis por definir sua estrutura de dados. É a partir do uso de atributos que será possível caracterizar (detalhar) as classes, sendo possível representar fielmente uma entidade do mundo real.

Há dois tipos principais de atributos:

* **Estáticos**: Mantém o mesmo valor durante sua existência, como a data de nascimento de uma pessoa;
* **Dinâmicos**: Valores que variam com o passar do tempo, como a idade de uma pessoa.

|  |
| --- |
| → Pegando o exemplo anterior de um restaurante, dentro da classe *Pessoa* podemos criar atributos para elas. Como:  – Tipo (cliente, funcionário), nome, idade, gênero, etc. |

## O Objeto

Já tínhamos visto que tudo começa com a definição de uma classe. Então o que é um objeto? É a instanciação de uma classe.

Um objeto é a representação de um conceito/entidade do mundo real, que pode ser física (bola, carro, árvore etc.) ou conceitual (viagem, estoque, compra etc.) e possui um significado bem definido para um determinado software. Para este conceito/entidade, deve ser definida inicialmente uma classe a partir da qual posteriormente serão instanciados objetos distintos.

Por definição, todo objeto é único. Se tivermos uma classe e forem criados dois objetos a partir dela, cada um será diferente do outro, mesmo que seus estados sejam iguais por coincidência.

Objetos tem características próprias (*atributos*) e executa ações (*métodos*), provenientes da classe que originou o objeto.

## A Mensagem

Mensagem é o processo de ativação de um método de um objeto. Isto ocorre quando uma requisição (chamada) a esse método é realizada, assim disparando a execução de seu comportamento descrito por sua classe. Pode também ser direcionada diretamente a classe, caso a requisição seja a um método estático.

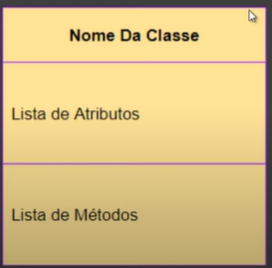
A definição anterior deixa bem claro que, quando requisitamos que um comportamento (código) de um método seja executado, estamos passando uma mensagem a este método. Mensagens podem ser trocadas entre métodos dos objetos/classes, para serem realizadas as atividades inerentes a cada um.

## Método

Método é uma lógica contida em uma classe para atribuir comportamentos (sequência de comandos), identificada por um nome. É muito similar a funções e procedimentos. O ato de invocar (chamar) um método é a passagem de mensagens para o objeto.

Uma função dentro da classe vai se chamar método!

## O que é UML?

É uma linguagem gráfica de modelagem de sistemas orientados a objetos. É um padrão mundialmente aceito. UML significa *Unified Modeling Language* (em português, Linguagem de Modelagem Unificada).

Ela provê várias notações gráficas para facilitar o processo de modelagem de sistemas OO. A forma padrão para representação de uma classe usando um diagrama de classes UML é:

## Herança

Herança é o relacionamento entre classes em que uma classe chamada de subclasse (classe filha, classe derivada) é uma extensão, um subtipo, de outra classe chamada de superclasse (classe pai, classe mãe, classe base). Devido a isto, a subclasse consegue reaproveitar os atributos e métodos dela. Além dos que venham a ser herdados, a subclasse pode definir seus próprios membros.

Em outras palavras, herança é um relacionamento entre classes, no qual uma classe “herda” os membros (atributos e métodos) de outra classe. Assim, podemos criar classes mais complexas sem repetição de código.

|  |
| --- |
| → Vamos criar um exemplo em que precisamos classificar mamíferos no reino animal.  → Vamos pegar os mamíferos: Homens, Baleias e Gatos. Esses compartilham características, mas possuem atributos distintos. Como poderíamos modelar eles?  → Podemos criar classes distintas para cada ser, mas essas classes teriam muitos comportamentos em comum, o que ocasionaria repetição desnecessária de código.  → A herança resolve esse problema:  Criamos uma classe **Mamífero** que possua as características comuns a todos os mamíferos, e então as classes Humano, Baleia e Gato herdando essas características e também implementando as funcionalidades específicas de cada animal, como Falar, Andar, Nadas e Miar.  A seta sempre aponta da classe ‘filho’(*que está herdando)* para a classe ‘pai’ (*que está com suas características sendo herdadas*). |

## Polimorfismo

Em determinados momentos em uma hierarquia de classes, precisamos que um mesmo método (nome e lista de parâmetro, ou seja, assinatura) se comporte de forma diferente dependendo do objeto instanciado a partir de uma classe de uma hierarquia qualquer.

Uma operação de um objeto pode assumir mais de um comportamento dependendo da chamada recebida, tratando e devolvendo respostas distintas.  
Polimorfismo ocorre quando um objeto tem um comportamento diferente para uma mesma ação.

***Poli =*** Muitos***; Morphos*** = Formas ***→*** Portanto, polimorfismo significa “muitas formas”.

|  |
| --- |
| → Exemplo: nas classes Gato e Humano há um comportamento (método) em comum: **Andar**.  → Porém, cada uma dessas classes implementa o método de forma distinta – gatos andam em quatro patas, e os humanos andam em duas pernas.  → Desta forma, dependendo de como um método é invocado, seu comportamento será diferente. |

## Definir uma classe em Python

A sintaxe simplificada para criar uma classe dentro é python é a seguinte:

|  |
| --- |
| Dentro da classe se pode então inicializar com o método \_\_init\_\_ |

### Introdução ao Método \_\_init\_\_

Esse método \_\_init\_\_, também conhecido como método especial, é o construtor da classe. Como o nome sugere, ele será o responsável por criar objetos a partir da classe em questão. Ou seja, sempre que for necessário criar objetos de uma determinada classe, seu construtor deverá ser utilizado.

Este método define o construtor da classe, como já foi falado, e nada mais é de onde definimos com uma nova instância será criada.

|  |
| --- |
| → Um exemplo para isso, é um registro de clientes em um restaurante. Onde cada cliente que entra precisa se registrar.  → Então trazendo pra python: toda vez que a classe Cliente é iniciada o \_\_init\_\_ faz seu papel para construir uma nova instância. |

#### Método \_\_init\_\_() de uma subclasse (herança)

Como já vimos anteriormente, uma herança é quando uma classe herda de outra, assumindo automaticamente todos os atributos e métodos da primeira classe.

A classe original se chama *classe-pai* e a nova é a *classe-filha.*

A primeira tarefa de Python ao criar uma instância de uma classe-filha é atribuir valores a todos os atributos da classe-pai. Para isso, o método \_\_init\_\_() de uma classe-filha precisa da ajuda de sua classe-pai.

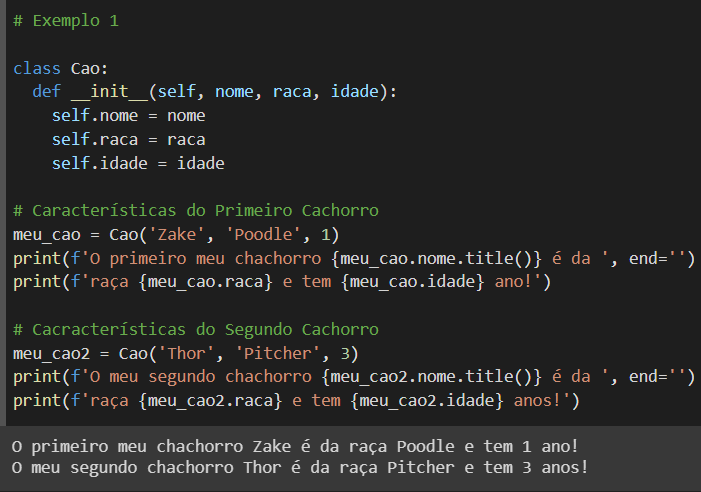
### Entendendo o self em classes

O uso do **self** é usado em classes no Python para indicar que você está referenciando alguma coisa do próprio objeto (sejam eles atributos ou métodos) – na verdade, o self é o próprio objeto em si.

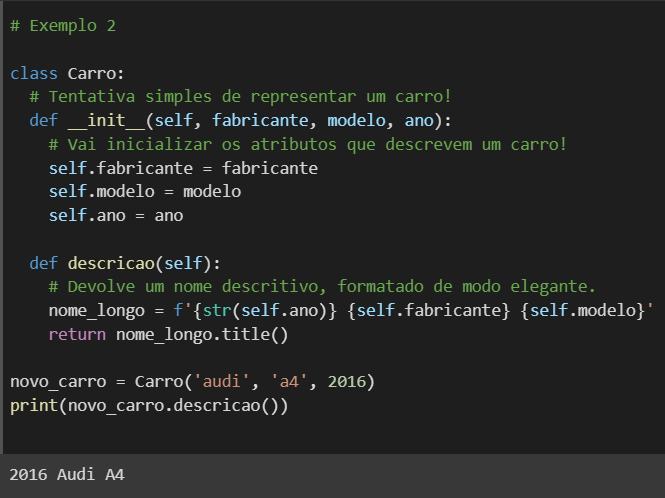
Em geral, quando criamos um construtor de uma classe (com o método \_\_init\_\_) colocamos o self nos atributos definidos ali para que eles sejam acessíveis nos métodos (raramente uma tributo que você inicializa no construtor não será usado em algum canto na classe).

Resumindo, todo método de uma classe recebe como primeiro parâmetro uma referência à instância que chama o método (isso é uma obrigação), permitindo assim que o objeto acesse os seus próprios atributos e métodos. Por convenção, chamamos esse primeiro parâmetro de self, mas qualquer outro nome poderia ser utilizado.

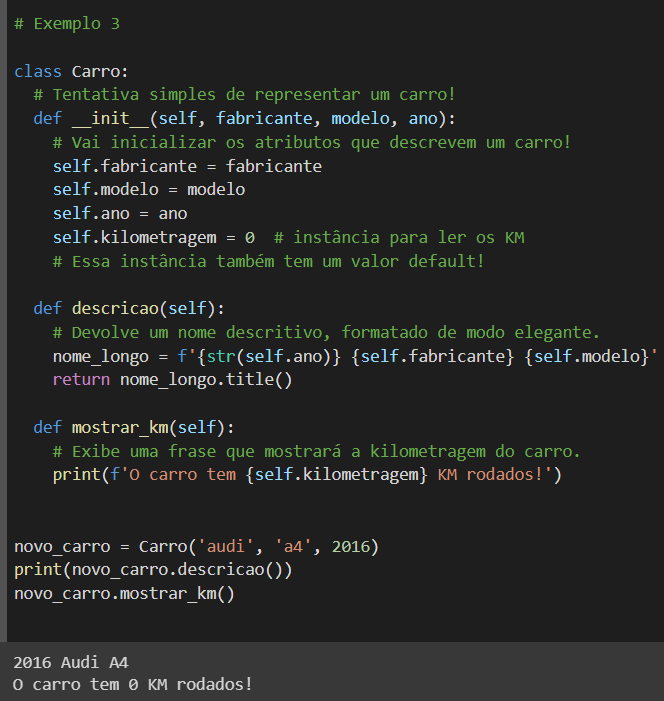
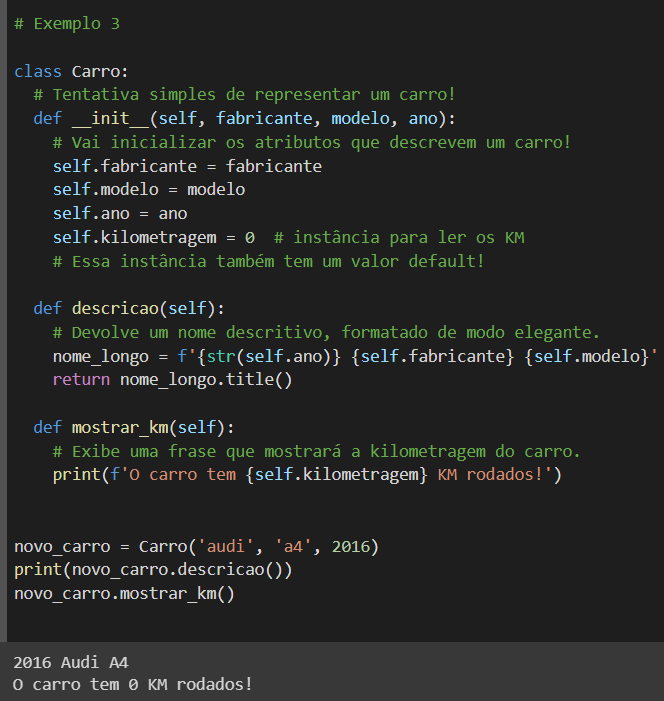
## Exemplos Práticos em Python

=> EX1: Para iniciar, criaremos uma classe simples que pegará informações de um cachorro e retornará nos prints uma frase.

=> EX2: Vamos escrever uma nova classe que represente um carro. Nossa classe armazenará informações sobre o tipo de carro com que estamos trabalhando e terá um método que sintetiza essa informação.

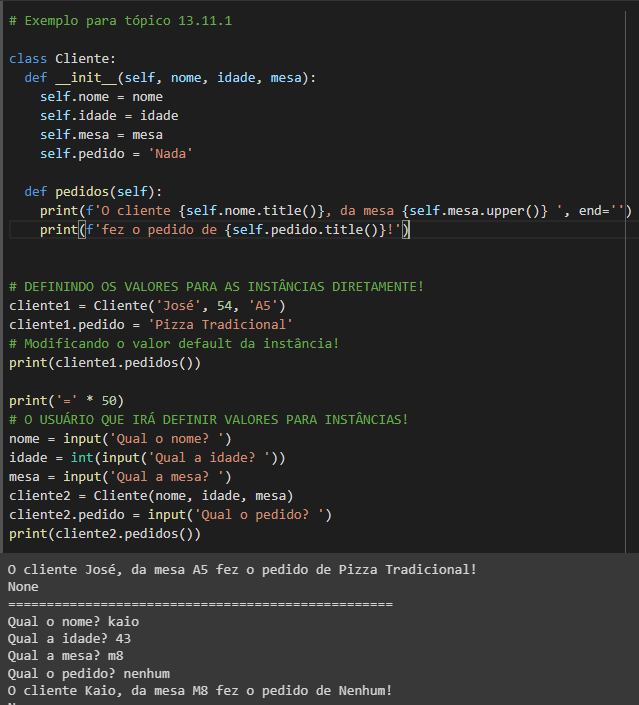


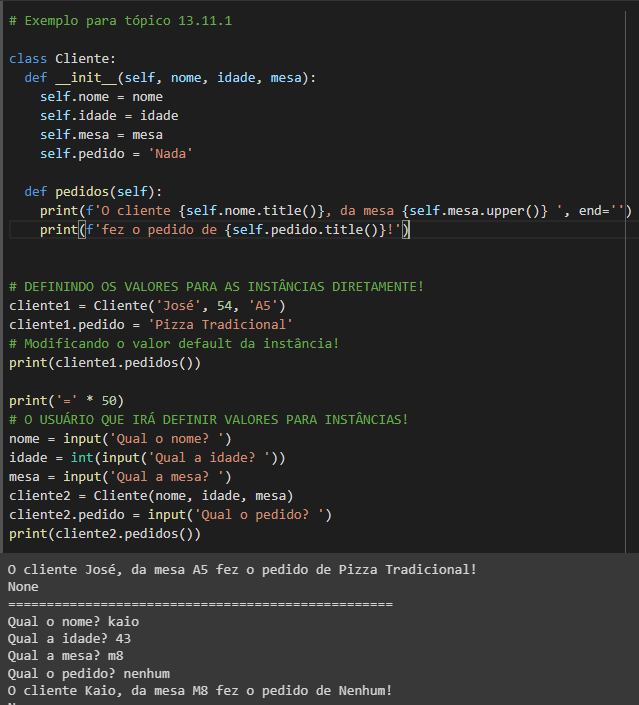
=> EX3: Vamos pegar o exemplo anterior de criar uma classe Carro, e deixar mais interessante, que é adicionar um atributo que mude com o tempo. Acrescentaremos um atributo que armazena a milhagem (KM) do carro.



### Modificar valores em instâncias

Como vimos no exemplo 3, criamos uma instância, chamada ‘kilometragem’ que recebe um valor default igual a 0. Esse valor também pode ser modificado para alguma coisa que o usuário desejar, ou que o próprio programador desejar usar.

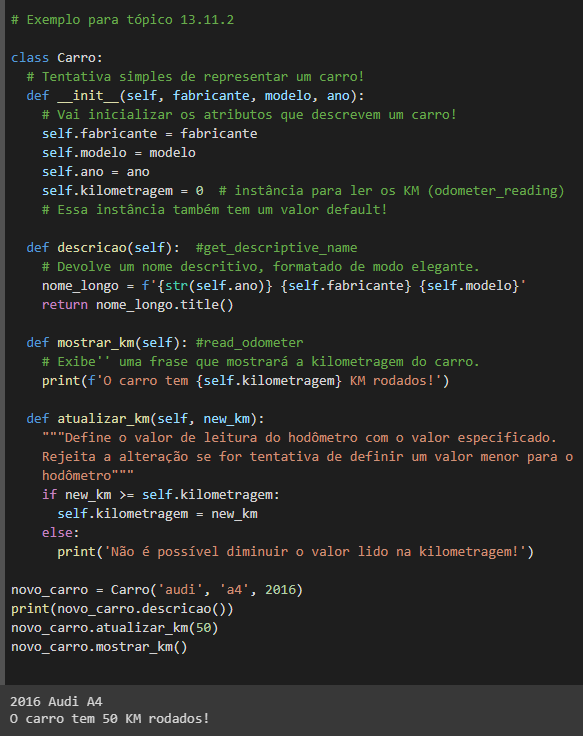
Como um método funciona quase igual a uma função, o método input também funciona normalmente. Vamos pegar um novo exemplo e modificar para vermos as duas maneiras de atribuir valores para as instâncias



### Modificando o valor de um atributo com um método

Pode ser conveniente ter métodos que atualizem determinados atributos para você. Em vez de acessar o atributo de modo direto, passe o novo valor para um método que trate a atualização internamente.

Vamos continuar utilizando o exemplo da Classe Carro (EX3):



Explicação:

Agora, atualizar\_km() verifica se o novo valor da kilometragem faz sentido antes de modificar o atributo.

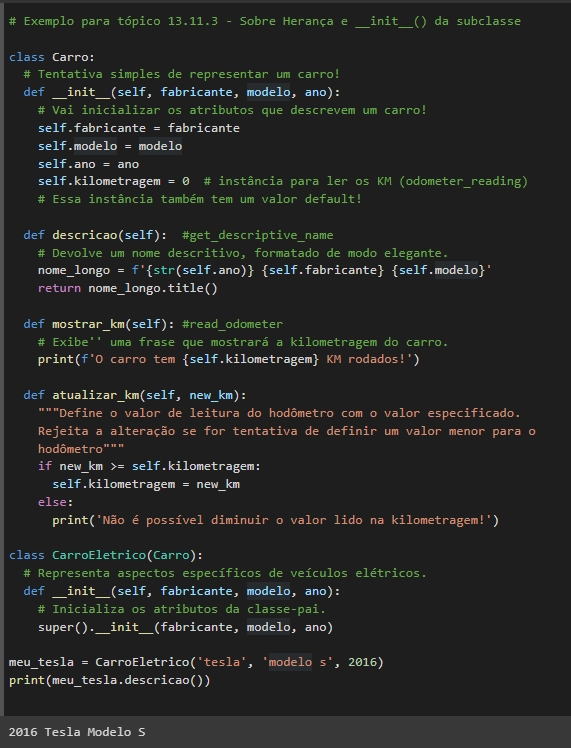
Se a nova milhagem for maior ou igual à milhagem existente, self.kilometragem, você poderá atualizar o valor de leitura da kilometragem com a nova milhagem.

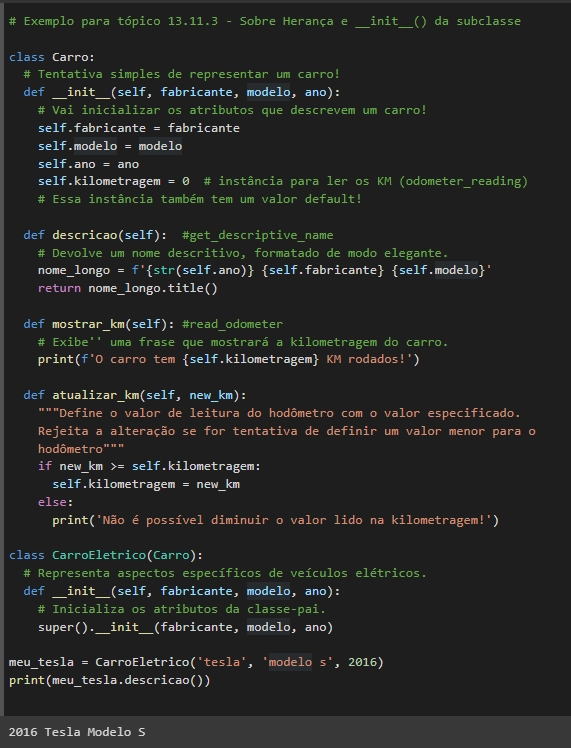
Se a nova milhagem for menor que a milhagem existente, você receberá um aviso informando que não pode diminuir o valor lido na kilometragem.

### Aprendendo a usar o método \_\_init\_\_ de uma classe filha

Vamos continuar nosso aprendizado utilizando os exemplos de carros. Dessa vez, vamos modelar um carro elétrico, e ele é apenas um tipo específico de carro, portanto podemos basear nossa nova classe **CarroEletrico** na classe **Carro** que escrevemos antes.

Vale relembrar também que, quando formos criar uma classe-filha (subclasse) temos que especificar entre parênteses na definição da classe-filha.





→ As partes iniciais são a mesma de antes. Quando criamos uma classe-filha, a classe-pai deve fazer parte do arquivo atual e deve aparecer antes da classe-filha no arquivo.

→ O método \_\_init\_\_() aceita as informações necessárias para criar uma instância de Carro.

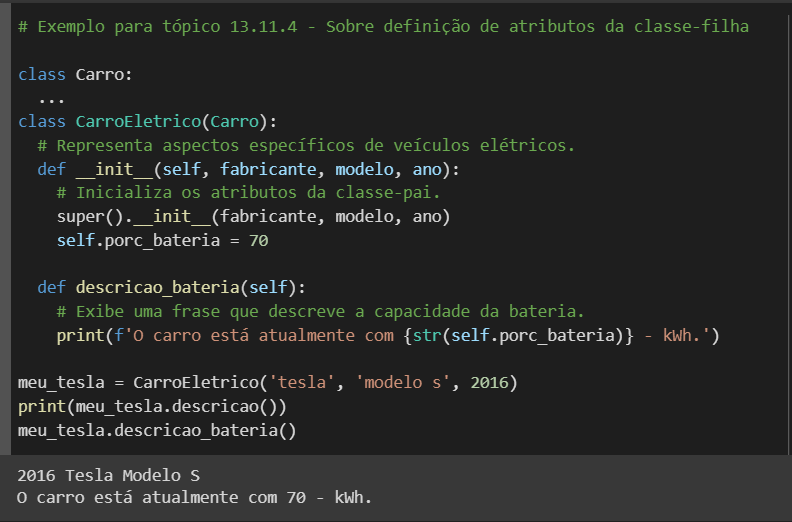
→ A função **super()** é uma função especial que ajuda Python a criar conexões entre a classe-pai e a classe-filha. Essa linha diz a python para chamar o método \_\_init\_\_() da classe-pai de CarroEletrico, que confere todos os atributos da classe-pai a CarroEletrico.  
→ O nome **super** é derivado de uma convenção segundo a qual a classe-pai se chama (superclasse) e a classe-filha é a subclasse.

### Definindo atributos e métodos da classe-filha

Depois que tiver uma classe-filha que herde de uma classe-pai, você pode adicionar qualquer atributo ou método novos necessários para diferenciar a classe-filha da classe-pai.

Vamos acrescentar um atributo que seja específico aos carros elétricos (uma bateria, por exemplo) e um método para mostrar esse atributo. Armazenaremos a capacidade da bateria e escreveremos um método que mostre uma descrição dela.

Obs.: Não vai ser colocado todo o código aqui novamente, apenas algumas partes e as partes novas.



# Arquivos

Aprender a trabalhar com arquivos e a salvar dados deixará seus programas mais fáceis de usar. Os usuários poderão escolher quais dados devem fornecer e quando. Aprender a tratar exceções também ajudará a lidar com situações em que os arquivos não existam e com outros problemas que possam fazer seus programas falharem.

Um arquivo de computador é um recurso de armazenamento de dados que está disponível em todo tipo de dispositivo computacional, seja um computador, dispositivo móvel, uma câmera fotográfica, entre outros.

***Nota***: verificar seção 7 do livro Python3 – Conceitos e aplicações, para melhor explanação do que é arquivo e seus tipos, além de revisar sobre arquivo binário que será importante na área de dados.

## Ler dados de um arquivo

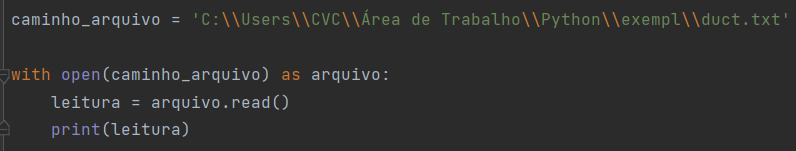
Ler dados de um arquivo é particularmente útil em aplicações de análise de dados, mas também se aplica a qualquer situação em que você queira analisar ou modificar informações armazenadas em um arquivo. Quando quiser trabalhar com as informações de um arquivo-texto, o primeiro passo será ler o arquivo em memória. Você pode ler todo o conteúdo de um arquivo ou pode trabalhar com uma linha de cada vez.

### Lendo um arquivo inteiro

Vamos começar criando um arquivo simples, contendo algumas linhas de texto. Esse arquivo será chamado ‘*duct.txt*’. Dentro desse arquivo está contido alguns nomes de pessoas:

|  |
| --- |
| José  Maria  márcia  Felipe  Jeniffer  Luiza  Pedro  john  faraday |

Agora vamos fazer um programa que nós permitirá fazer a leitura de todo o arquivo de uma única vez.

Explicação:

A variável ‘*caminho\_arquivo*’ é puramente para armazenar a localização de onde está o arquivo, caso o arquivo estivesse na mesma localização do programa (leitor.py) não precisaria pegar todo o caminho e poderia simplesmente colocar o nome do arquivo (como vamos ver mais a frente).

A palavra reservada **with** fecha o arquivo depois que não for mais necessário acessá-lo. Dá para trocar por close(), mas se um bug em seu programa impedir que a instrução close() seja executada, o arquivo não será fechado.

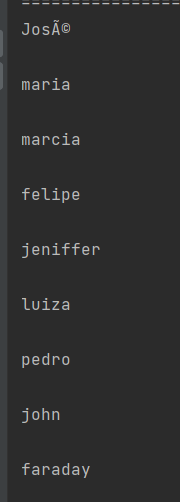
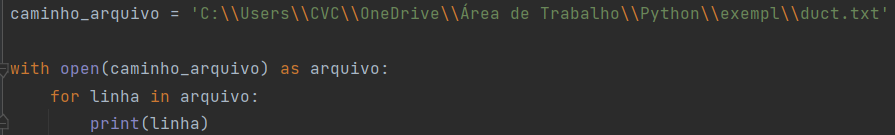
Para realizar qualquer tarefa com um arquivo, mesmo que seja apenas exibir seu conteúdo, você precisará inicialmente *abrir* o arquivo para acessá-lo. A função **open()** é justamente isso que faz, abre o arquivo. Ela precisa de um argumento dentro dos parênteses: o nome do arquivo que você quer abrir.   
E no caso, pode ser passado tanto como uma variável que recebe o caminho (como o exemplo), ou pode ser passado diretamente dentro dos parênteses.

O método read() é basicamente para ler todo o conteúdo do arquivo e armazená-lo em uma longa string em ‘*leitura*’.

### Lendo dados linha a linha

Quando estiver lendo um arquivo, com frequência você vai querer analisar cada linha do arquivo. Talvez você esteja procurando determinada informação no arquivo ou queira modificar o texto do arquivo de alguma maneira.

Para isso, podemos usar um laço for no arquivo para analisar cada uma de suas linhas, uma de cada vez.

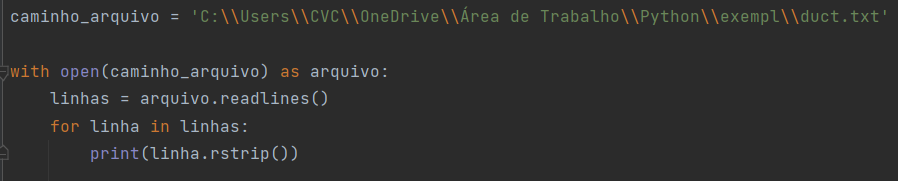
Saída:

A saída do nosso programa vai resultar com algumas linhas em branco que não contém no arquivo original. Essas linhas em branco aparecem porque um caractere invisível de quebra de linha está no final de cada linha do arquivo-texto. A instrução print adiciona a sua própria quebra de linha sempre que a chamamos, portanto acabamos com dois caracteres de quebra de linha no final de cada linha: um do arquivo e outro da instrução print. Se usarmos rstrip() em cada linha na instrução print, eliminamos essas linhas em brancos extras.

### Criando uma lista de linhas de um arquivo

Quando usamos o **with**, o objeto arquivo devolvido por open() (que é a variável) estará disponível somente no bloco with que o contém. Se quiser preservar o acesso ao conteúdo de um arquivo fora do bloco with, você pode armazenar as linhas do arquivo em uma lista dentro do bloco e então trabalhar com essa lista.

Pode processar partes do arquivo imediatamente e postergar parte do processamento de modo que seja feito mais tarde no programa.

O método **readlines()** armazena cada linha do arquivo em uma lista. Essa lista é então armazenada em **linhas**, com a qual podemos continuar trabalhando depois que o bloco with terminar.

Usamos um laço for simples para exibir cada linha de **linhas**. Como cada item de **linhas** corresponde a uma linha do arquivo, a saída será exatamente igual ao conteúdo do arquivo.

## Caminho de arquivo no Windows

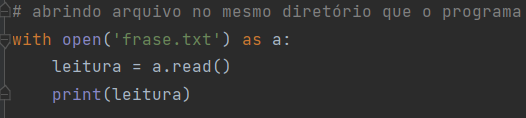
Quando um nome de arquivo simples é passado para a função open(), Python observa o diretório em que o arquivo executado no momento está armazenado. Mas, às vezes, o arquivo que você quer abrir não estará no mesmo diretório que o seu arquivo de programa.

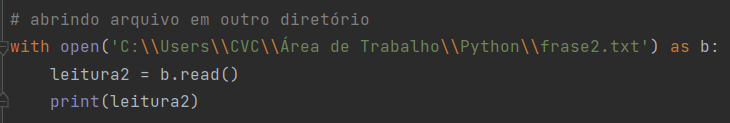
Para fazer Python abrir arquivos de um diretório que não seja aquele em que seu arquivo de programa está armazenado, é preciso fornecer um path de arquivo (caminho), que diz a python para procurar em um local específico de seu sistema.

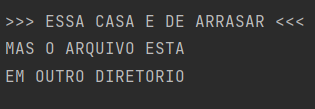
No linux e no mac as coisas são fáceis, apenas copiar e colar o caminho já resolve. Mas no Windows não é a mesma coisa. Em sistemas Windows, use duas barras invertidas (\\) para separar as pastas.

Explicando no exemplo é mais fácil de entender:

O arquivo atual (test.py) está na pasta cap10







## Modos de manipulação de arquivo

Como já vimos, para abrir um arquivo com Python é necessário a utilização da função *open()*. E vimos também que a função precisa de parâmetros, o parâmetro obrigatório que já vimos é basicamente o nome do arquivo.

Mas também existem outros parâmetros para se colocar entre os parênteses. Vamos olha para a sintaxe: *open(‘nome’, ‘modo’, ‘buffering’)*. O parâmetro ‘modo’ é a forma de abertura do arquivo, que é o que iremos ver aqui na seção. O buffering é uma coisa que não precisamos nos preocupar muito no início, visto que esse parâmetro é a quantidade de bytes reservados na memória para a abertura do arquivo, e ele pode ser omitido, assim como o modo.

Os modos podem variar, sendo só leitura, só escrita ou os dois juntos. Os valores mais comumente utilizados para parâmetro ‘modo’ são:

|  |  |
| --- | --- |
| r | – Somente para leitura |
| w | – Escrita (*caso o arquivo já exista, ele será apagado e um novo arquivo será criado*) |
| x | – Criação (*Permite a criação do arquivo exclusivamente se este não existir. Caso exista, levanta a exceção ‘FileExistsError’* |
| a | – Leitura e Escrita ao final (*adiciona o novo conteúdo ao fim do arquivo*) (é como um append() |
| b | – Modo binário |
| t | – Modo Texto |
| + | – Leitura e Escrita (*permite adicionar leitura e escrita no arquivo. Independente de qual seja o modo*. |
| – r+ → leitura e escrita |
| – w+ → Escrita (o modo w+, assim como o w, apaga o conteúdo anterior do arquivo) |
| – a+ → Leitura e escrita (*a diferença é que ele abre o arquivo p/ atualização* ) |

## Método úteis para arquivos

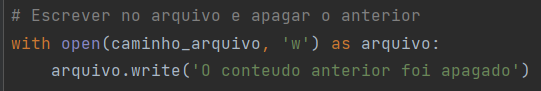
Para os exemplos dos métodos, utilizaremos um arquivo chamado ‘arquivoteste.txt’ que conterá, inicialmente, uma frase. E ao decorrer do método vamos mudando.

|  |
| --- |
| ‘Olá Mundo!’ ‘Vamos continuar observando!’ |

### Método write (e writelines)

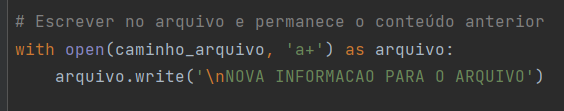
Às vezes, você pode querer excluir um conteúdo do arquivo e substituí-lo totalmente por um novo conteúdo. Podemos fazer isso utilizando o método write().

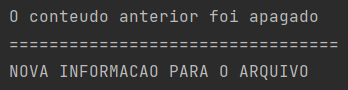
**Escrever no arquivo e apagar o anterior:**



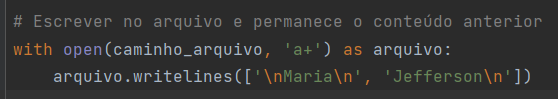


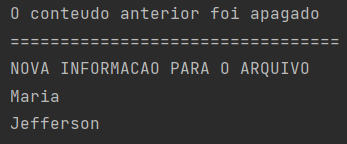
**Escrever no arquivo e deixar o conteúdo anterior:**

****



O método **writelines()** vai servir para escrever várias linhas de uma única vez.



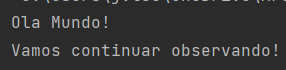


### Método read, readline e readlines

Os métodos para ler um arquivo são simples e que já vimos nos tópicos anteriores. Para lermos um arquivo, precisamos ter uma forma de ‘interagir’ com ele em nosso programa e é exatamente isso que métodos fazem.

#### Read()

O primeiro método que precisamos aprender retorna todo o conteúdo do arquivo com uma string.

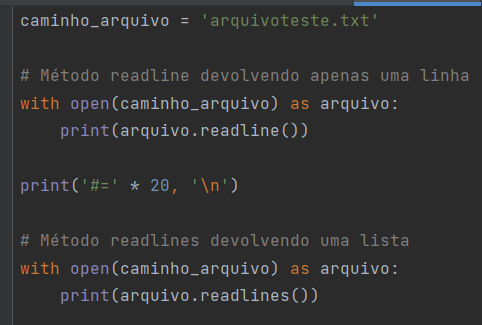


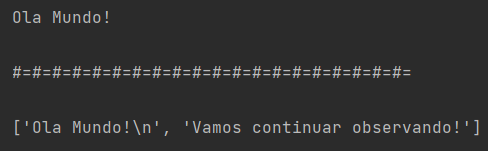
#### Readline() x Readlines()

Podemos ler o arquivo linha por linha através desses dois métodos. Mas eles têm uma pequena diferença.

**Readline()** lê uma linha do arquivo até o fim dessa linha.  
É possível, como opção, passar o tamanho, o número máximo de caracteres que desejamos incluir no resultado da string.

Já **Readlines()** retorna uma lista com todas as linhas do arquivo como elementos individuais (strings).

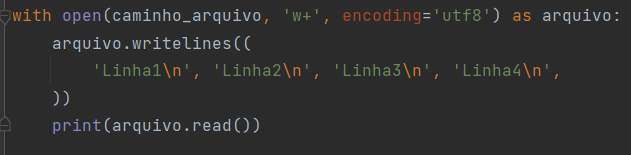




### Método seek

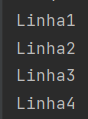
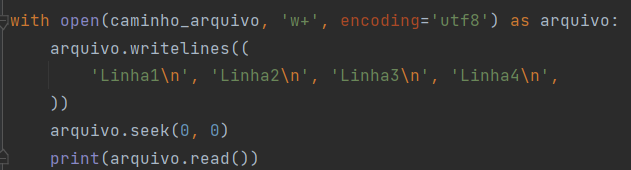
A função seek() é usada para alterar a posição do identificador de arquivo (é como um ‘cursor’) para uma determinada posição. O identificador de arquivo define de onde os dados devem ser lidos ou gravados no arquivo.

Então imagine um arquivo com 4 linhas, o identificador estará no final dela, se eu mandar ler, ele vai começar a ler do final, por que o identificador estará no fim.





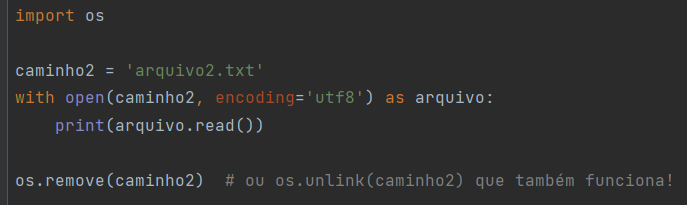
Agora utilizaremos a função **seek()** para determinar onde ficará o identificador, que no caso será no inicio da linha:



### Excluir arquivos

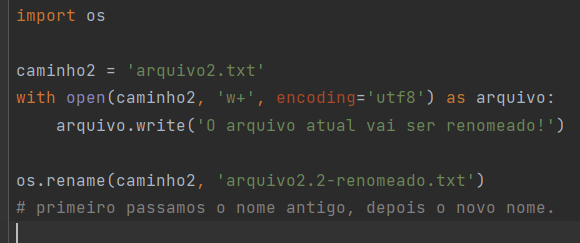
Para excluir arquivos utilizando Python, é preciso importar um módulo chamado **os**, que contém as funções para interagir com o seu sistema operacional.

Dentro do módulo existe a função remove() ou unlink() que recebe o caminho do arquivo como argumento e o exclui automaticamente.



### Renomear arquivos

Dentro do módulo **os** também é possível utilizar uma função para renomear ou mover o arquivo. E essa função é o **rename()**.



Como usamos uma variável para armazenar o local de onde está o arquivo, ou simplesmente o nome do arquivo, dá para movermos também utilizando o rename, que no caso, seria apenas colocar no novo nome, o local de onde quer mover.

### Enconding

# Exceções

Python usa objetos especiais chamados exceções para administrar erros que surgirem durante a execução de um programa. Sempre que ocorrer um erro que faça Python não ter certeza do que deve fazer em seguida, um objeto exceção será criado. Se você escrever um código que trate a exceção, o programa continuará executando.

Se a exceção não for tratada, o programa será interrompido e um traceback, que inclui uma informação sobre a exceção levantada, será exibido. As exceções são tratadas com blocos try-except. Um bloco try-except pede que Python faça algo, mas também lhe diz o que deve ser feito se uma exceção for levantada.

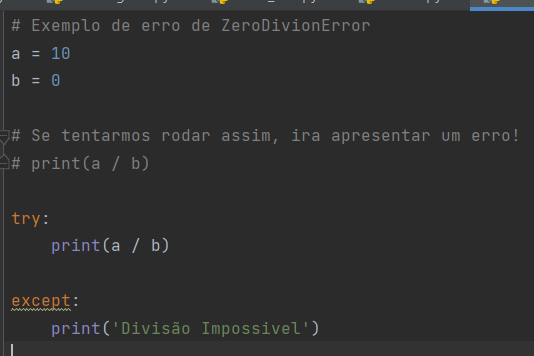
Ao usar blocos try-except, seus programas continuarão a executar, mesmo que algo comece a dar errado.

## Try, except, eles e finally

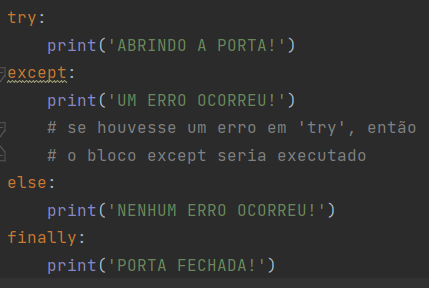
* **try**: Bloco de código a ser executado;
* **except** {exceção}:
  + Código que será executado caso a exceção seja capturada;
  + A exceção pode ser omitida.
* **else**:
  + Código que será executado caso nenhuma exceção tenha sido lançada ou capturada;
* **finally**:
  + Código que será executado independente se alguma exceção for capturada ou não.

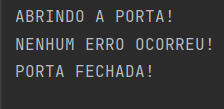
Apenas as cláusulas **try** e **except** são obrigatórias, sendo **else** e **finally** opcionais! Além disso, é possível ter múltiplas cláusulas **except**, capturando exceções diferentes.

Vamos observar um erro simples, e ele é: A **exceção ZeroDivisionError:**



Vamos a outro **exemplo:**



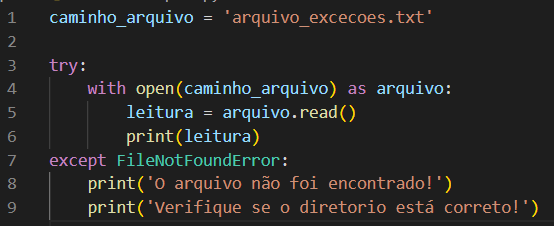


Python também disponibiliza uma lista com algumas das exceções, sendo uma das maiores a exceção ‘Exeption’ e as outras, como *ZeroDivionError*, uma sub-exceção.  
Para acessar todas, basta ir no link: https://docs.python.org/pt-br/3/library/exceptions.html

## Tratando exceções para arquivos

### FileNotFoundError

Um problema comum ao trabalhar com arquivos é o tratamento de arquivos ausentes. O arquivo que você está procurando pode estar em outro lugar, o nome do arquivo pode estar escrito de forma incorreta ou o arquivo talvez simplesmente não exista. Podemos tratar todas essas situações de um modo simples com um bloco try-except.



– O arquivo não está criado!



### Criando e lendo um arquivo com exceções

Python → modulo 6 → exer115 → arquivo → \_\_init\_\_.py

# Trabalhando com dados (tipos de dados)

## JSON – JavaScript Object Notation

+ informações: Udemy: aula - 289

Haverá casos em que precisaremos armazenar tipos de dados, e as vezes nós teremos que pedir aos usuários que forneçam determinados tipos de informação.

Qualquer que seja o foco de seu programa, você armazenará as informações fornecidas pelos usuários em estruturas de dados como listas e dicionários.

Quando os usuários fecham um programa, quase sempre você vai querer salvar as informações que eles forneceram. Uma maneira simples de fazer isso envolve armazenar seus dados usando o módulo json.

O módulo json permite descarregar estruturas de dados Python simples em um arquivo e carregar os dados desse arquivo na próxima vez que o programa executar. Também podemos usar json para compartilhar dados entre diferentes programas Python.

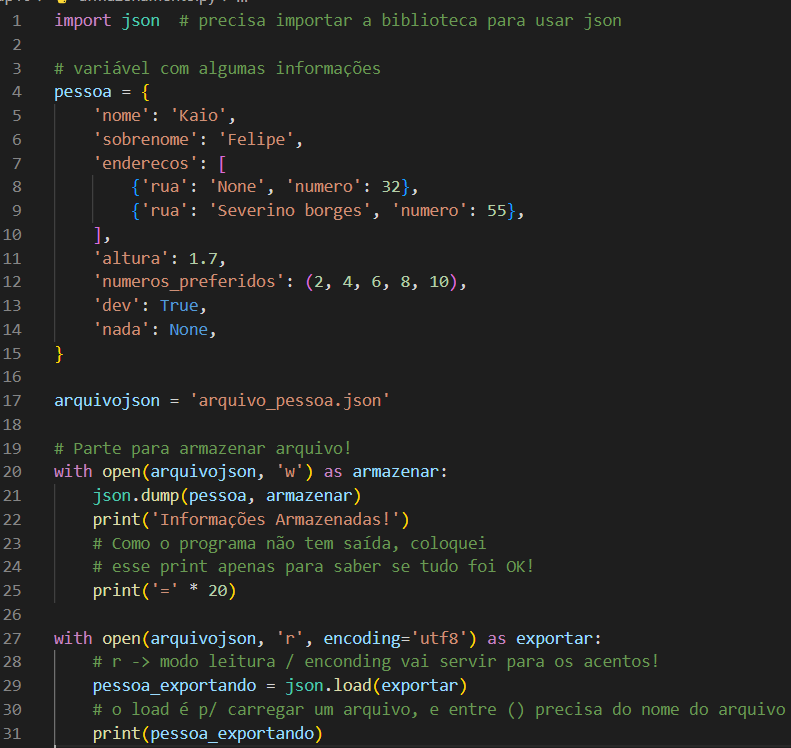
**NOTA:** O formato JSON (JavaScript Object Notation, ou Notação de Objetos JavaScript) foi originalmente desenvolvido para JavaScript. Apesar disso, tornou-se um formato comum, usado por muitas linguagens, incluindo Python.

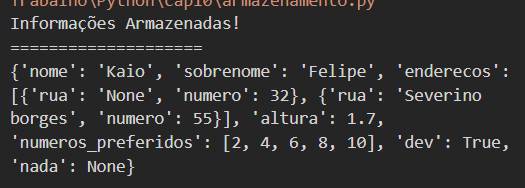
**NOTA2:** O formato JSON é muito simples, ela não suporta coisas que executam ações específicas como, por exemplo, funções, métodos, ou classes. Além de outras coisas.

### Utilizando json.dump() e json.load()

Para o exemplo, criarei uma variável que armazenará algumas informações sobre uma pessoa. Dentro dessa variável vai conter dicionários, listas, valores inteiros e booleanos.

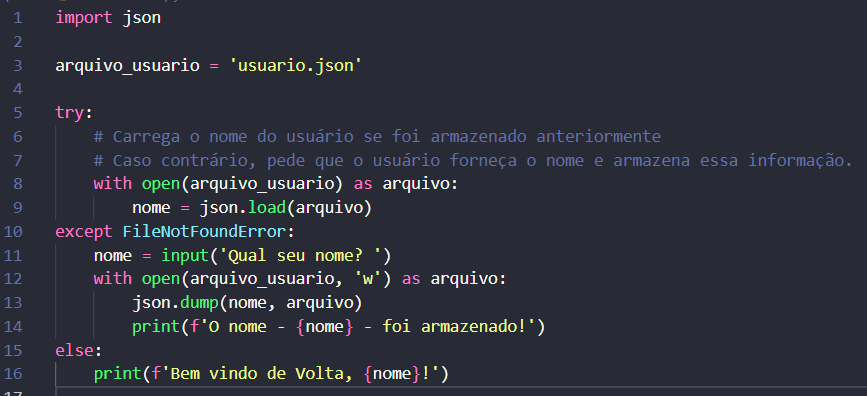
E nesse programa, ele armazena as informações e logo depois ele traz de volta.





### Salvando um dado gerado por usuário

Salvar dados com json é conveniente quando trabalhamos com dados gerados pelo usuário porque, se você não salvar as informações de seus usuários de algum modo, elas serão perdidas quando o programa parar de executar. Vamos observar um exemplo em que pedimos aos usuários que forneçam seus nomes na primeira vez em que o programa executar e, então, o programa deverá lembrar esses nomes quando for executado novamente.

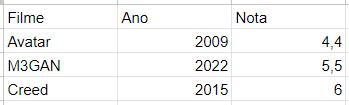


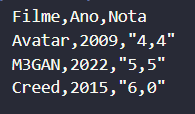
## CSV – Comma Separated Values

CSV ou Valores separados por vírgulas é um formato de arquivo que armazena dados em forma de tabela, onde cada linha representa uma linha da tabela e as colunas são separadas por vírgulas. Ele é amplamente utilizado para transferir dados entre sistemas de diferentes plataformas, como por exemplo, para importar ou exportar dados para uma planilha (*Google Sheets, Excel, LibreOffice Calc*) ou para uma base de dados.

Um arquivo CSV geralmente tem a extensão “**.csv**” e pode ser aberto em um editor de texto ou em uma planilha eletrônica.

Vamos a um exemplo de arquivo CSV. Nele teremos três colunas e três linhas:

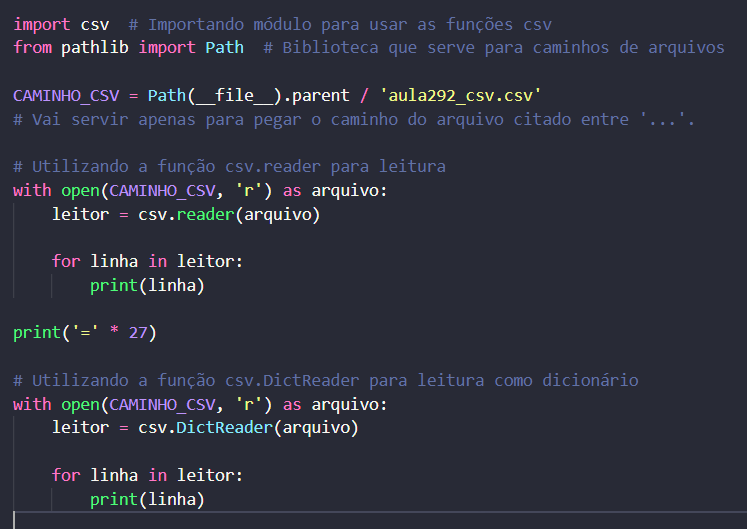
Informações aberto no **Google Sheets**

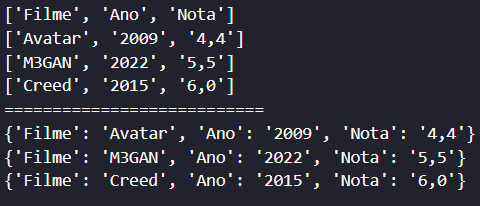
Informações aberto no **VSC**

É importante comentar que, para separar uma coluna de outra utilizasse de um delimitador que é a **vírgula(,)**. Mas como foi feito no exemplo, utilizou-se as vírgulas entre as notas, e para permitir isso, coloca as *aspas* (podendo ser ‘ ou “) entre as informações.

Outra regra importante do CSV é a de que não pode haver e nem deixar linhas ou espaços sobrando dentro do arquivo.

### Como ler o CSV no Python





### Como escrever em arquivos CSV no Python

# Referências

* **Métodos comuns para strings→**
  + <http://devfuria.com.br/python/strings/>
  + <https://blog.betrybe.com/python/python-split/>
  + <https://www.acervolima.com.br/2021/01/python-string-split.html>
  + https://www.pythonprogressivo.net/2018/10/String-Maiuscula-Minuscula.html
  + http://ptcomputador.com/P/python-programming/93783.html
  + https://acervolima.com/string-capitalize-em-python/
  + https://acervolima.com/metodo-python-string-count-1/
  + https://pt.stackoverflow.com/questions/249532/como-manipular-strings-com-find
* **Alinhamento de F-Strings →**
  + [https://www.hashtagtreinamentos.com/f-strings-em-python#:~:text=As%20f%2Dstrings%20v%C3%A3o%20servir,vari%C3%A1vel%20dentro%20de%20%7B%7D%20chaves.&text=D%C3%A1%20s%C3%B3%20uma%20olhada%20como,informa%C3%A7%C3%B5es%20dentro%20de%20um%20texto](https://www.hashtagtreinamentos.com/f-strings-em-python" \l ":~:text=As%20f-strings%20v%C3%A3o%20servir,vari%C3%A1vel%20dentro%20de%20%7B%7D%20chaves.&text=D%C3%A1%20s%C3%B3%20uma%20olhada%20como,informa%C3%A7%C3%B5es%20dentro%20de%20um%20texto).
  + https://acervolima.com/alinhamento-de-string-em-f-string-python/
* **Fatiamento de Strings →**
  + https://pense-python.caravela.club/08-strings/04-fatiamento-de-strings.html
* **Utilizações em Listas →**
  + [https://www.freecodecamp.org/portuguese/news/o-metodo-sort-para-arrays-e-listas-em-python-ordenacoes-ascendente-e-descendente-explicadas-e-com-exemplos/#:~:text=O%20m%C3%A9todo%20sort()%20permite,valor%20intermedi%C3%A1rio%20para%20cada%20elemento](https://www.freecodecamp.org/portuguese/news/o-metodo-sort-para-arrays-e-listas-em-python-ordenacoes-ascendente-e-descendente-explicadas-e-com-exemplos/" \l ":~:text=O%20m%C3%A9todo%20sort()%20permite,valor%20intermedi%C3%A1rio%20para%20cada%20elemento).
* **Utilizações em Tuplas →**
* **Utilizações em Dicionários →** 
  + [https://pythonacademy.com.br/blog/dicts-ou-dicionarios-no-python#:~:text=chave3'%3A%200%7D-,Operador%20de%20%E2%80%9Cdesempacotamento%E2%80%9D,e%20em%20nossos%20queridos%20dicion%C3%A1rios](https://pythonacademy.com.br/blog/dicts-ou-dicionarios-no-python" \l ":~:text=chave3'%3A%200%7D-,Operador%20de%20%E2%80%9Cdesempacotamento%E2%80%9D,e%20em%20nossos%20queridos%20dicion%C3%A1rios).
* **Módulos em Python →**
* **Cores em Python →**
* **Orientação a Objetos →**
  + https://www.youtube.com/watch?v=dG7LlYne2VA&list=PLucm8g\_ezqNqj--UUSn16yfDp3xcZi40t&index=2
  + https://cursos.alura.com.br/forum/topico-uso-do-self-136352
  + https://panda.ime.usp.br/aulasPython/static/aulasPython/aula17.html
* **Arquivos e Exceções →**
  + [https://diegomariano.com/manipulando-arquivos/#:~:text=Para%20abrir%20um%20arquivo%20de,abertura%20do%20arquivo%20(opcional](https://diegomariano.com/manipulando-arquivos/" \l ":~:text=Para%20abrir%20um%20arquivo%20de,abertura%20do%20arquivo%20(opcional)).
  + https://www.freecodecamp.org/portuguese/news/como-escrever-em-um-arquivo-em-python-open-read-append-e-outras-funcoes-de-manipulacao-explicadas/