Python para Data Science: funções, estruturas de dados e exceções

Tags

_Bibliotecas do python

> Conjunto de módulos e funções úteis para a pessoa usuária

_Pip

Gerenciador de bibliotecas do Python

_Organiza as bibliotecas e pacotes dentro do Python

Já instalamos a biblioteca agora vamos importar a biblioteca

Para saber a versão do matplotlib

matplotlib.**version**

Para saber mais: PIP e PyPI

O PIP é um **gerenciador de bibliotecas** para Python. Nele temos acesso a todas as bibliotecas já instaladas em nossa aplicação junto com a versão de cada uma delas. A partir das linhas de comando, podemos instalar, atualizar e remover pacotes Python facilmente. Com o *pip*, podemos facilmente instalar pacotes de terceiros em projetos Python.

Para conseguirmos acessar todos os pacotes instalados em nosso Jupyter Notebook, no Colab, podemos escrever o seguinte código:

```
# Imprimindo todos os pacotes instalados no ambiente e suas versões
!pip list
```

O PIP funciona conectando-se ao Python Package Index (PyPI) que é o maior repositório centralizado para pacotes Python com milhares de bibliotecas disponíveis para instalação. Podemos pesquisar no PyPI para encontrar pacotes que atendam às necessidades e, em seguida, usar o pip para instalá-los em nossos projetos.

O PyPI é mantido pela Python Software Foundation e contém uma ampla variedade de pacotes Python. Nele são publicados pacotes Python para que outras pessoas desenvolvedoras possam usá-los. Logo, o PIP e o PyPI são duas ferramentas importantes em Python.

Versões em bibliotecas

Aprendemos que as bibliotecas possuem versões. Assim, selecione a alternativa que descreve corretamente uma razão pela qual alguém possa querer instalar uma versão específica de uma biblioteca e demonstra um exemplo com sua sintaxe utilizando o gerenciador de pacotes pip no ambiente do Google Colab.

💡 A instalação de uma versão específica de uma biblioteca pode decorrer da necessidade do uso de recursos específicos para determinados projetos ou tarefas a serem executadas.
!pip install pandas==1.5.0

Algumas versões de uma biblioteca podem ter recursos que são necessários para um determinado projeto ou tarefa. Por exemplo, a versão 1.5.0 do Pandas pode ter um recurso específico que não está disponível em versões mais recentes ou mais antigas.

_Biblioteca random()

Gerar números em sequências aleatórias

(https://docs.python.org/pt-br/3/library/random.html)

Importar uma função específica de uma biblioteca

```
# Importando uma função específica de uma biblioteca
from nome_biblioteca import metodo

# Importando uma função específica de uma biblioteca
from random import choices
```

Dica: Você pode notar ao longo de nossa prática a importância de recorrer a documentação para aprender como utilizar um método ou pacote na linguagem Python.

O método `help()`, por exemplo, retorna uma descrição sobre uma variável, método ou classe.

https://docs.python.org/pt-br/3/library/functions.html?#help

Para saber mais: outras formas de importação

Já trabalhamos com duas formas de importação de pacotes: o `import nome_biblioteca` para todo o pacote e o `from nome_biblioteca import metodo` para apenas um método de uma dada biblioteca.

A importação de métodos específicos de uma biblioteca pode trazer algumas vantagens para o nosso projeto como:

- **Economia de memória**: quando importamos uma biblioteca inteira, todo o seu código é carregado na memória, mesmo que não utilizemos todos os seus métodos. Importar apenas os métodos que precisamos pode economizar memória, especialmente em programas com grandes bibliotecas.
- **Maior clareza no código**: importar apenas os métodos que vamos usar torna o código mais claro e fácil de entender.
- **Redução de conflitos de nome**: quando importamos uma biblioteca inteira, podemos acabar tendo conflitos de nome com outras variáveis ou funções em nosso código.
- **Redução no tempo de execução**: como a biblioteca inteira não é importada, o tempo de execução do programa pode ser mais rápido, uma vez que menos código precisa ser carregado na memória e interpretado pelo interpretador Python.

Além das formas vistas anteriormente, podemos citar mais dois exemplos que você poderá encontrar ao longo de suas práticas e estudos da linguagem Python:

`from nome_biblioteca import met_1, met_2`

Este código resulta na importação de 2 ou mais métodos de uma biblioteca, não necessitando repetir a importação desta a cada método desejado. Podemos, por exemplo, importar 2 métodos da biblioteca random para colher uma amostra de 5 valores de uma lista de 20 valores gerada aleatoriamente com números de 0 a 99.

```
from random import randrange, sample

lista = []

for i in range(0, 20):
  lista.append(randrange(100))

sample(lista, 5)COPIAR CÓDIGO
```

> Saída: [28, 66, 53, 81, 85]

`from nome_biblioteca import *`

Esta forma é utilizada para importar todos os métodos de uma dada biblioteca. A diferença desta para o `import nome_biblioteca` é que, neste caso, não precisamos usar o nome da biblioteca para chamar um método. Podemos passar apenas o nome do método.

Por exemplo, se formos calcular a raiz quadrada de um certo número poderíamos seguir por uma das duas formas:

Usando `import nome_biblioteca`

```
import math

n = int(input("Digite um número positivo para calcular sua raiz quadrada:"))
print(f"\nA raiz quadrada de {n} é igual a {math.sqrt(n)}")
```

Usando `from nome_biblioteca import *`

```
from math import *

n = int(input("Digite um número positivo para calcular sua raiz quadrada:"))
print(f"\nA raiz quadrada de {n} é igual a {sqrt(n)}")
```

Note que, no segundo exemplo, suprimimos o nome `math` utilizando o método desejado e escrevendo o código com menos caracteres.

> Atenção: A importação nesse sentido precisa de alguns cuidados:
>
> - Podemos ter choque de nomes entre as variáveis. Por exemplo, no caso de termos uma função chamada `sqrt` antes de importar a da biblioteca `math`.
> - Podemos reduzir a eficiência da execução, se o número de funções importadas é grande.
> - Não fica explícito de onde aquela variável, método ou classe veio.

Quanto eu vendi?

Uma empresa quer calcular o número de itens vendidos a partir da receita da venda e do valor da unidade do produto. Além disso, precisa arredondar para cima o número de itens vendidos caso o valor não seja um número inteiro.

Suponha que a receita foi de 1000 reais e a unidade do produto é vendida a 15 reais. Quais dos códigos abaixo utilizam corretamente o método `ceil` da biblioteca `math` para realizar esse arredondamento?

> Você pode recorrer a documentação do método ceil.

```
from math import ceil

receita = 1000
unidade = 15

print(f" A empresa vendeu aproximadamente {ceil(receita/unidade)} unidades")`
```

Esta é uma forma correta de utilizar um método de uma dada biblioteca. Como importamos apenas um método específico da biblioteca `math`, para utilizar o método `ceil` escrevemos ele sem necessitar da referência da biblioteca de onde ele veio.

Desafio: hora da prática

Vamos praticar o que aprendemos até aqui solucionando os problemas propostos em código.

Aquecimento

1) Escreva um código para instalar a versão `3.7.1` da biblioteca `matplotlib`.

2) Escreva um código para importar a biblioteca `numpy` com o alias `np`.

3) Crie um programa que lê a seguinte lista de números e escolha um número desta aleatoriamente.

```
lista = [8, 12, 54, 23, 43, 1, 90, 87, 105, 77]
```

4) Crie um programa que sorteia aleatoriamente um número inteiro menor que 100.

> Dica: use a função randrange() da biblioteca random. Essa função recebe como parâmetro o valor limite para a escolha aleatória ou um intervalo se passado o limite mínimo e máximo. Por exemplo, randrange(5) gera valores inteiros menores que 5.

5) Crie um programa que solicite à pessoa usuária digitar dois números inteiros e calcular a potência do 1º número elevado ao 2º.

> Dica: use a função pow() da biblioteca math

Aplicando a projetos

6) Um programa deve ser escrito para sortear uma pessoa seguidora de uma rede social para ganhar um prêmio. A lista de participantes é numerada e devemos escolher aleatoriamente um número de acordo com a quantidade de participantes.

Peça à pessoa usuária para fornecer o número de participantes do sorteio e devolva para ela o número sorteado.

7) Você recebeu uma demanda para gerar números de token para o acesso ao aplicativo de uma empresa. O token precisa ser par e variar de 1000 até 9998. Escreva um código que solicita à pessoa usuária o seu nome e exibe uma mensagem junto a esse token gerado aleatoriamente.

`"Olá, [nome], o seu token de acesso é [token]! Seja bem-vindo(a)!"`

8) Para diversificar e atrair novos clientes, uma lanchonete criou um item misterioso em seu cardápio chamado "salada de frutas surpresa". Neste item, são escolhidas aleatoriamente 3 frutas de uma lista de 12 para compor a salada de frutas da pessoa cliente.

Crie o código que faça essa seleção aleatória de acordo com a lista abaixo:

```
frutas = ["maçã", "banana", "uva", "pêra",
          "manga", "coco", "melancia", "mamão",
          "laranja", "abacaxi", "kiwi", "ameixa"]
```

9) Você recebeu um desafio de calcular a raiz quadrada de uma lista de números, identificando quais resultaram em um número inteiro. A lista é a seguinte:

```
numeros = [2, 8, 15, 23, 91, 112, 256]
```

Informe no final quais números possuem raízes inteiras e seus respectivos valores.

> Dica: use a comparação entre a divisão inteira (//) da raiz por 1 com o valor da raiz para verificar se o número é inteiro. Por exemplo:
>
> ```
> num = 1.5
> num_2 = 2
> print(f'{num} é inteiro? :', num // 1 == num)
> print(f'{num_2} é inteiro? :', num_2 // 1 == num_2)
> ```

Saída:

```
1.5 é inteiro? : False
2 é inteiro? : True
```

10) Faça um programa para uma loja que vende grama para jardins. Essa loja trabalha com jardins circulares e o preço do metro quadrado da grama é de R$ 25,00. Peça à pessoa usuária o raio da área circular e devolva o valor em reais do quanto precisará pagar.

> Dica: use a variável pi e o método pow() da biblioteca math. O cálculo da área de um círculo é de: A = π*r^2 (lê-se pi vezes raio ao quadrado).

Caso precise de ajuda, opções de solução das atividades estarão disponíveis na seção “Opinião da pessoa instrutora”.

> OBS: Para te ajudar a verificar seus códigos, deixo disponibilizado um notebook dos desafios para você construir suas soluções. Você pode baixar ele para resolver as atividades e desafios testando os seus conhecimentos até o momento.

_Funções >

Na linguagem Python, as **funções** são sequências de instruções que executam tarefas específicas, podendo ser reutilizadas em diferentes partes do código. Elas podem receber parâmetros de entrada (que podemos chamar de *inputs*) e também retornar resultados.

- Sequências de instruções
- Podem ser reutilizadas em diferentes partes do código.
- Recebem **parâmetros** de entrada **(inputs)**
- Podem **retornar valores**

_Built - In Functions >

O interpretador do Python já possui uma série de funções embutidas que podem ser invocadas a qualquer momento. Algumas que vamos utilizar ao longo desse curso são: type(), print(), list(), zip(), sum(), map() etc.

São funções embutidas

Podem ser **invocadas a qualquer momento**

Exemplos: **print(), type(), list()**

Arredondar a média usando round():

https://docs.python.org/pt-br/3/library/functions.html#round

help(round)

round?

O que são as built-in functions?

As **built-in functions**, ou funções embutidas, fornecem uma série de funcionalidades básicas para a aplicação de tarefas e instruções na linguagem Python. Elas são muito úteis em Data Science pela facilidade de uso e abstração em tarefas que podem ser complexas, como ordenações de lista ou manipulação de strings.

Sabendo disso, qual das opções abaixo descreve corretamente o que são as built-in functions no Python e outras vantagens na sua utilização?

As built-in functions são funções integradas ao Python, ou seja, já estão disponíveis no interpretador sem necessidade de importação. Uma de suas vantagens é a eficiência.

As built-in functions são funções integradas ao Python que já estão disponíveis e não possuem a necessidade de importação, o que torna sua utilização muito mais fácil e eficiente.

Tipos de built-in function

O `round()` e `pow()` são built-ins de manipulação de dados. O `round()` retorna o arredondamento de números passando a quantidade de dígitos desejada, já o `pow()` calcula a potência da base e expoente passados.

Ambas as funções são responsáveis pelo tratamento matemático dos dados recebidos, sendo o `round()` utilizado quando queremos arredondar um número para o inteiro mais próximo ou para um número especificado de casas decimais. E o `pow()` que é equivalente ao operador `**` calcula a potência a partir de uma base e expoente passada pela pessoa usuária.

Funções no contexto de dados

As **funções personalizadas** são uma das principais ferramentas de programação em Python. Elas permitem que as pessoas programadoras ou cientistas de dados definam blocos de código reutilizáveis, tornando o código mais legível, organizado e fácil de manter. Elas podem ser usadas para uma ampla variedade de tarefas, desde cálculos matemáticos simples até a implementação de algoritmos complexos sendo essenciais para a criação de código eficiente e flexível.

Falando de funções personalizadas para um contexto de ciências de dados, quais das alternativas abaixo demonstram algumas das suas vantagens e como podem ser utilizadas?

A utilização de funções personalizadas em ciências de dados possibilita a modularização do código criando um script a ser seguido tornando-o mais organizado e fácil de manter. Uma aplicação prática seria utilizar funções para carregar diferentes fontes de dados como arquivos CSV e bancos de dados para serem lidos pelo código e, em seguida, analisados pela pessoa usuária.

A criação e utilização de funções em Python possibilita a separação do código em partes menores e mais organizadas, facilitando a manutenção e entendimento do mesmo. Isso é especialmente útil em projetos extensos e complexos de ciências de dados. Além disso, no exemplo citado, é bem frequente utilizar funções para carregar dados de diferentes fontes e pré-processar esses dados.

A utilização de funções personalizadas em ciências de dados permite a reutilização do código, evitando repetições desnecessárias de processos. Uma aplicação prática seria utilizar funções para limpar e pré-processar dados, como remover valores ausentes e padronizar o formato de datas, economizando tempo e evitando erros comuns.

A reutilização do código é uma das principais vantagens de se criar funções. Elas podem ser utilizadas em diferentes partes do código, evitando repetições desnecessárias e tornando o código mais eficiente e legível. Além disso, no exemplo citado, a criação de funções para limpeza de dados e criação de gráficos pode facilitar bastante o trabalho em ciências de dados, permitindo que esses processos sejam repetidos de maneira rápida e eficiente.

A utilização de funções personalizadas em ciências de dados pode reduzir o tempo de execução do código, resultando em maior eficiência computacional. Uma aplicação prática seria utilizar funções para criar gráficos e visualizações de dados. Isso pode tornar a exploração dos dados mais fácil e rápida.

A utilização de funções não tem um impacto direto na eficiência computacional do código. A eficiência é determinada pela forma como o código é implementado, não pelo uso de funções. No entanto, no exemplo citado, a criação de funções para gerar gráficos e visualizações de dados é bem interessante e pode auxiliar na exploração de padrões e tendências nos dados.

Para saber mais: escopo de uma variável

Em Python, o escopo de uma variável é definido pela região do código onde ela pode ser acessada. No caso de uma função, o escopo pode ser dividido em duas categorias: **escopo global** e **escopo local**.

O escopo global é o espaço no qual uma variável pode ser acessada por qualquer função ou código que esteja sendo executado no programa. Já o escopo local é o espaço no qual a variável pode ser acessada apenas pela função em que foi definida.

O problema de escopo ocorre quando uma variável é definida dentro do escopo de uma função e, em seguida, é referenciada fora do escopo da função. Nesse caso, o Python gera uma mensagem de erro, indicando que a variável não foi definida (`NameError`).

Abaixo segue um exemplo que ilustra esse comportamento. Vamos inicialmente criar uma variável `x` externa a função `soma()`, na qual vamos definir uma outra variável y e, por fim, imprimir a soma das duas variáveis.

```
x = 7

def soma():
  y = 9
  print(x + y)
```

Note que o `x` é a nossa variável definida no escopo global e o `y` a variável definida no escopo local da função `soma()`. Quando tentamos executar a nossa função, a soma é realizada normalmente:

```
soma()
```

Saída:

```
16
```

No entanto, quando tentamos imprimir a soma de `x` e `y` fora do escopo da função, o Python gera um erro, pois a variável `y` existe apenas dentro da função `soma()`.

```
print(x + y)
```

Saída:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-f09a7b03ddbf> in <module>
----> 1 print(x + y)

NameError: name 'y' is not defined
```

Para corrigir esse erro, podemos tornar `y` uma variável global ou retornar seu valor na função e atribuí-lo a uma variável externa. No próximo vídeo, vamos aprender como proceder nesse tipo de situação.

Faça como eu fiz: documentando funções

É importante deixar o nosso código ou análise dos dados o mais acessível possível para o público. E, uma das formas de atingir esse propósito reside na documentação de funções.

Neste tópico, podemos auxiliar quem lê o nosso projeto ou utiliza as funções que desenvolvemos a entender quais tipos de variáveis podemos utilizar, se existem ou não valores padrões ou até descrever de maneira sucinta o que aquele pedaço de código faz.

Aqui, pedimos que você siga esse passo a passo na documentação da função de média que construímos durante a aula. Ela será novamente requisitada mais a frente em nossos estudos.

Type Hint

O Type Hint é uma sintaxe utilizada no Python para indicar o tipo de dados esperado de um parâmetro ou retorno de função, auxiliando na legibilidade e manutenção do código. Podemos dizer em poucas palavras que ela seria as dicas de tipagem dos dados.

Formato:

```
def <nome>(<param>: <tipo_param>) -> <tipo_retorno>:
  <instruções>
  return resultadoCOPIAR CÓDIGO
```

Trazendo para o nosso exemplo da função `media()`, podemos utilizar o type hint da seguinte forma:

```
# a nossa função recebe uma lista do tipo list e retorna uma variável do tipo float
def media(lista: list) -> float:
  calculo = sum(lista) / len(lista)
  return calculoCOPIAR CÓDIGO
```

Se escrevermos a função `media()` em outra célula e passarmos o mouse por cima, podemos observar a seguinte imagem apontando as dicas de tipagem dos parâmetros de entrada e retorno da função:

Default value

No Python, o Default Value é um valor padrão atribuído a um argumento de função, que é utilizado caso nenhum valor seja passado pelo usuário.

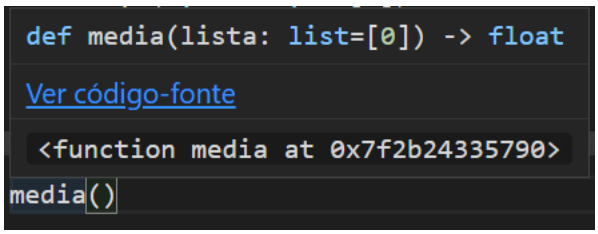
Formato:

```
  <nome_variavel>: <tipo_variavel> = <valor_variavel>
```

Implementando ainda mais a nossa função `media()`, podemos utilizar o default value da seguinte forma:

```
# a nossa função recebe uma lista do tipo list e retorna uma variável do tipo float
# caso não recebe nenhum valor de parâmetro será passada uma lista com um único
# elemento sendo ele zero
def media(lista: list=[0]) -> float:
  calculo = sum(lista) / len(lista)
  return calculoCOPIAR CÓDIGO
```

Da mesma forma que fizemos no type hint, se escrevermos a função `media()` em outra célula e passarmos o mouse por cima, podemos observar o que está na imagem abaixo. Nela temos apontado tanto as dicas de tipagem dos parâmetros de entrada e retorno da função quanto valor padrão para nossa lista, caso a pessoa usuária não passe nenhum valor na execução.



Se executarmos a função média, essa será a saída:

Saída:

```
0.0
```

Docstring

Por fim, temos o Docstring que é uma string literal usada para documentar um módulo, função, classe ou método em Python. Ela é colocada como o primeiro item de definição e pode ser acessada usando a função help().

O Docstring deve descrever o propósito, parâmetros, tipo de retorno e exceções levantadas pela função. É uma boa prática de programação utilizar Docstrings em seu código para facilitar a leitura, manutenção e compartilhamento do código com outras pessoas desenvolvedoras.

Formato:

```
def <nome>(<param_1>, <param_2>, ..., <param_n>):
    ''' Texto documentando sua função...
    '''
  <instruções>
  return resultadoCOPIAR CÓDIGO
```

Concluindo a implementação da nossa função `media()`, podemos utilizar o docstring da seguinte forma:

```
def media(lista: list=[0]) -> float:
  ''' Função para calcular a média de notas passadas por uma lista

  lista: list, default [0]
    Lista com as notas para calcular a média
  return = calculo: float
    Média calculada
  '''
  calculo = sum(lista) / len(lista)
  return calculoCOPIAR CÓDIGO
```

Se executarmos o código `help(media)` em uma outra célula, temos a seguinte saída:

Saída:

```
Help on function media in module __main__:

media(lista: list = [0]) -> float
    Função para calcular a média de notas passadas por uma lista

    lista: list, default [0]
      Lista com as notas para calcular a média
    return = calculo: float
      Média calculada
```

Vamos deixar o link de um artigo, em inglês, explicando alguns conceitos de documentação de códigos em Python.

_Função LAMBDA>

- Também chamada de **função anônima**
- **Não precisa** ser definida
- Pode ser **descrita** em **uma única linha**

Atrelar a função lambda com a função **map**

função map: **é uma built function que mapeia valores**

E quando ele é usado dentro do lambda, pode usar funções simples e funções rápidas

no meu dado e aplicar mapeamento de valores, então vou conseguir alterar valores de variáveis e também valores de estrutura de dados como listas, dicionários, e tudo isso usando o lambda.

```
map(<lambda function>, <iterador>)

# Notas do(a) estudante
notas = [6.0, 7.0, 9.0, 5.5, 8.0]
qualitativo = 0.5

notas_atualizadas = lambda x: x + qualitativo
notas_atualizadas(notas)
```

```
# Retornou um erro - Dizendo que a gente só consegue concatenar uma lista com uma outra lista
# o código está tentando ler o parâmetro "x" como uma lista completa e não como cada elemento da lista e o parâmetro qualitativo é um float
# ao ínves da variável somar 0.5 com cada elemento contido na lista. Está tentando somar na lista o valor 0.5.
# não consigo somar um valor em uma lista, o que eu consigo somar é os elementos contidos na lista.
# Está sendo interpretado / lido a lista completa e não cada elemento contido na lista
```

Para ajustar isso é necessário utilizar o map()

casos de estrutura de dados é necessário utilizar o map

```
# Não conseguimos aplicar o lambda em listas direto, é necessário
# utilizarmos junto a ela a função map
notas_atualizadas = map(lambda x: x + qualitativo, notas)
notas_atualizadas

map(função lambda, estrutura de dados que vamos usar para mapear os valores dentro dela)

a função map () cria um objeto - exemplo do caso de cima  -> <map at 0x7acb5dc1aaa0>

Quando as funções forem mapeadas vamos usar uma segunda built function chamada list()
porque a função list() transforma o objeto que foi criado com a função map() em uma lista
dessa forma conseguimos visualizar os valores mapeados
```

Convertendo temperaturas

Na aula de programação, a aluna Millena recebeu uma tarefa para converter uma lista de temperaturas de graus Celsius para Fahrenheit. Pesquisando sobre como realizar a conversão ela notou que era possível criar uma função lambda para converter as temperaturas, mas precisaria utilizar a função embutida `map()` para auxiliá-la na conversão elemento a elemento.

A conversão de temperaturas Celsius em Fahrenheit pode ser descrita da seguinte forma:

```
temp_fahrenheit = temp_celsius * 9/5 + 32COPIAR CÓDIGO
```

Qual das seguintes alternativas a Millena pode construir o seu mapeamento de valores gerando uma lista de temperaturas em graus Fahrenheit (`temp_fahrenheit`) a partir de uma lista de temperaturas em graus Celsius (`temp_celsius`) e exibir a lista gerada no console?

```
temp_celsius = [0, 25, 37, 78, 100]
temp_fahrenheit = list(map(lambda x: (x * 9/5) + 32, temp_celsius))
temp_fahrenheit

A função map() é usada aqui para aplicar a fórmula de conversão de Celsius para Fahrenheit
em cada elemento da lista temp_celsius, e a função lambda é usada para definir a fórmula.
```

Desafio: hora da prática

_Lista >

> Coleção ordenada de elementos que podem ser ou não do mesmo tipo

Lista de listas

Formato padrão:

```
[[a1, a2,...,an], [b1, b2,...,bn], ..., [n1, n2,...,nn]]
```

Importância da lista de listas

Avançando nos estudos em Python, nos deparamos com a necessidade de criação de estruturas mais robustas e complexas para salvarmos os dados de diversos tipos e analisarmos utilizando as técnicas de exploração dos dados. É bastante comum no dia a dia da pessoa cientista de dados trabalhar com estruturas de dados aninhadas como as listas de listas.

Dito isto, quais das alternativas abaixo apontam a importância de se trabalhar com listas de listas em um projeto de data science em Python?

- Listas de listas têm sua importância por permitirem armazenar dados em uma estrutura hierárquica.

  Elas permitem armazenar dados em uma estrutura hierárquica, o que pode ser útil em muitas situações em que os dados possuem várias dimensões.

Listas de listas são importantes, pois possibilitam o armazenamento de informação de maneira organizada, como em tabelas e matrizes.

Essa é uma das razões para que as listas de listas sejam amplamente utilizadas em ciência de dados. Por dividir os dados de forma organizada e semelhante ao formato matricial, a coleta e análise de dados pode ser aplicada seguindo padrões de leitura de dados.

A importância das listas de listas está na facilidade de manipulação de dados de diferentes tipos, separados em listas dentro da estrutura.

Elas podem ser usadas para armazenar dados de diferentes tipos e formatos, tornando o código mais flexível e adaptável a diferentes cenários de análise de dados.

Para saber mais: trabalhando com tuplas

_Tuplas>

Estrutura de dados usada para armazenar itesn em uma única variável

São imutáveis, não pode sofrer alteração e nem adição

Interessante caso for trabalhar com dados fixos

As **tuplas** são estruturas de dados imutáveis da linguagem Python que são utilizadas para armazenar conjuntos de múltiplos itens e frequentemente são aplicadas para agrupar dados que não devem ser modificados. Ou seja, não é possível adicionar, alterar ou remover seus elementos depois de criadas. Em nosso curso, vamos explorar um pouco mais desse tipo de estrutura voltada à aplicação em ciência de dados.

Tuplas são especialmente úteis em situações nas quais precisamos garantir que os dados não sejam alterados acidentalmente ou intencionalmente. Por exemplo, em um conjunto de dados que representa o cadastro de estudantes, podemos utilizar uma tupla para representar aquele(a) estudante em específico e manter no banco de dados de uma instituição de ensino. Dessa forma, garantimos que as informações de cada estudante não sejam alteradas inadvertidamente.

Para criar uma tupla, basta separar seus elementos por vírgulas e envolvê-los entre parênteses. Por exemplo, podemos criar uma tupla com um registro de uma estudante da seguinte maneira:

```
cadastro = ("Júlia", 23, "São Paulo", "SP", "Python para DS 1")
```

Para acessar os elementos de uma tupla, podemos usar o índice entre colchetes. Por exemplo:

```
print(cadastro[0]) # imprime Júlia
print(cadastro[-1]) # imprime Python para DS 1COPIAR CÓDIGO
```

Além disso, por também ser um iterável, podemos desempacotar os dados de uma tupla passando cada valor para uma variável. Por exemplo:

```
nome, idade, cidade, estado, turma = cadastro
```

E exibir os dados cadastrais da estudante:

```
print(f'A estudante {nome} tem {idade} anos e mora em {cidade}-{estado}. Ela está matriculada na turma de {turma}.')
```

Saída:

```
A estudante Júlia tem 23 anos e mora em São Paulo-SP. Ela está m
```

Achei interessante o código com tuplas

```
codigo_estudante = []

# Observação - Strings também são interáveis e posso acessar não só as palavras contidas nas listas, ou seja os valores, como também
# Eu posso acessar as letras

for i in range(len(estudantes)):
  codigo_estudante.append((estudantes[i], estudantes[i][0] + gera_codigo()))

codigo_estudante
```

Usos de listas de tuplas

Outra estrutura de dados composta e bastante utilizada no contexto de ciência de dados são as **listas de tuplas**. Diferente das listas de listas, elas são indicadas em situações nas quais precisamos garantir que os dados não sejam alterados acidentalmente ou intencionalmente, mas que possam ser acessados para agregar mais dados nas análises.

Dito isto, qual das alternativas abaixo representa um uso adequado da estrutura de listas de tuplas?

```
Um exemplo de listas de tuplas pode ser a de um conjunto de dados que contém informações sobre filmes, incluindo o título, ano de la

filmes = [ ('O Poderoso Chefão', 1972,'Drama'), ('O Senhor dos Anéis: O Retorno do Rei', 2003, 'Aventura'), ('Interestelar', 2014, '
```

Este exemplo pode ser adequado para uma lista de tuplas, uma vez que filme, ano de lançamento e gênero são previamente definidos e não necessitam de atualização nos itens da tupla. Além disso, a lista de tuplas pode receber novos registros (tuplas), adicionando-os dentro da lista.

```
Um exemplo de listas de tuplas pode ser a de uma lista de presença de uma
instituição de ensino sendo preenchida por um formulário. Por exemplo:

lista_presenca = [(1, 'Maria Clara'), (2, 'Antonio Costa'), (3, 'Joana Dias'), (4, 'Marcelo Souza')]
```

Este exemplo pode ser adequado para uma lista de tuplas, visto que o formulário pode receber novos registros (tuplas). Quem sofrerá a atualização será a lista recebendo novas tuplas.

List comprehension

É uma forma simples e concisa de criar uma lista. Podemos aplicar condicionais e laços para criar diversos tipos de listas a partir de padrões que desejamos para a nossa estrutura de dados.

https://docs.python.org/pt-br/3/tutorial/datastructures.html?#list-comprehensions

Formato padrão:

```
[exressão for item in lista]
```

Serve para você construir uma lista no python de forma mais rápida.

Função no Python

Recomendação

- Faça o código primeiro e depois coloque ele dentro de uma **função !**

```
# Imprime numerado
for i, linguagem in enumerate(minha_lista):
    print(i + 1, "=>", linguagem)
```

Saída:

1 => ruby

2 => python

3 => lua

4 => javascript

5 => go

6 => C++

A função enumerate() retorna uma tupla de dois elementos a cada iteração: um número sequencial e um item da sequência correspondente.

def funcao: exemplo Parâmetro (*numero)

Dentro de uma função, significa que não se sabe a quantidade de parâmetros suficientes de parâmetros e pode ficar passando vários parâmetros.

```
def tamanho(*num):
    tam = len(num)
    return print(tam)

def contador(*num):
  for valor in num:
    print(f'valor: {valor}')


contador(5, 10, 15)

Outro exemplo

def dobra(lst):
    pos = 0
    while pos <= lst:
        lst[pos] *= 2
        pos += 1


Outro teste que acabei fazendo

def entrada():
  s = 0
  n = int(input('O valor de entrada '))
  s = n + n
  return print(f'Soma do número: {s}')

entrada()
```

Outra Função

```
def titulo(msg):
  print("-" * len(msg))
  print(msg)
  print("-" * len(msg))
```

Funções - pt 2

em questão de parâmetros da função

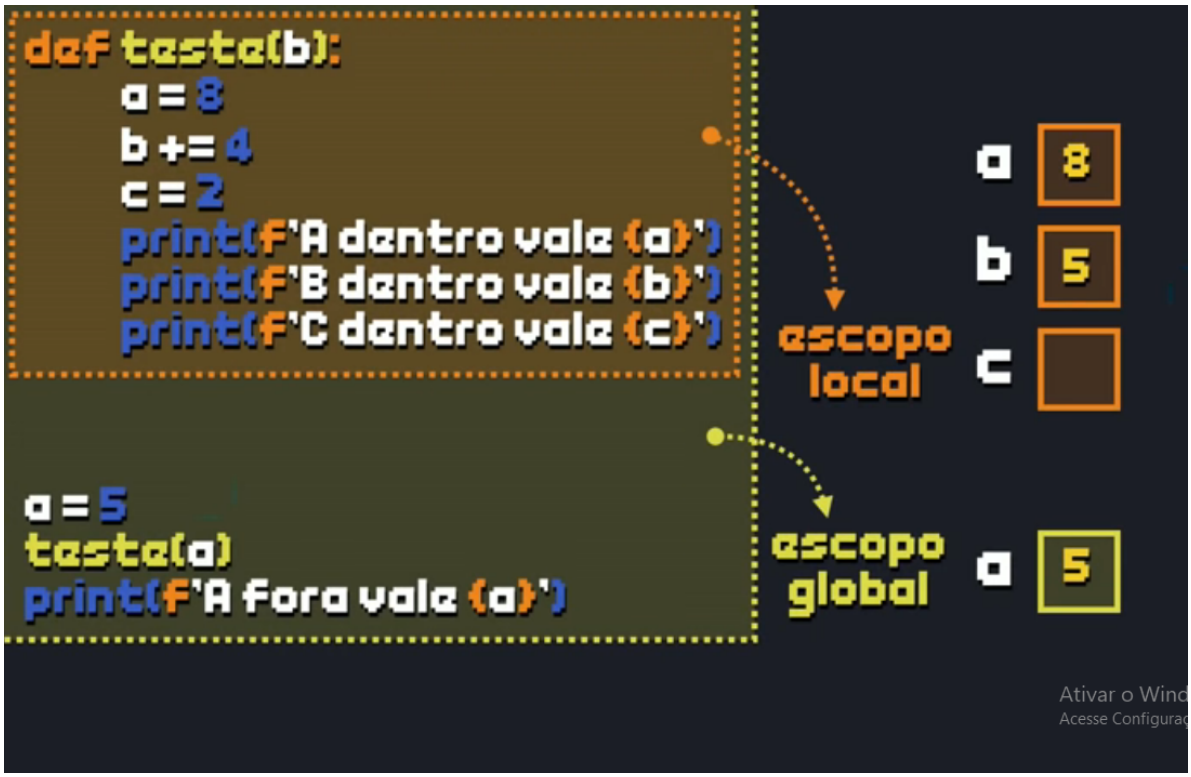
```
def somar(a, b, c):
    s = a + b + c
    print(f'A soma é >> {s}')

somar(5, 5, 5)
```

```
  resultado vai ser 15

somar(5, 5)
  vai retornar um erro

  para contornar isso, eu posso colocar que os meus parâmetros recebem um valor
  padrão

def somar(a, b, c = 0):
    s = a + b + c
    print(f'A soma é >> {s}')


somar(5, 5)
  não vai mais gerar erro, porque o valor padrão vai ser 0

  ou todos os parâmetros com valor padrão
```

Escopo do Python

uma variável global **a**

Outra variável Local **a**

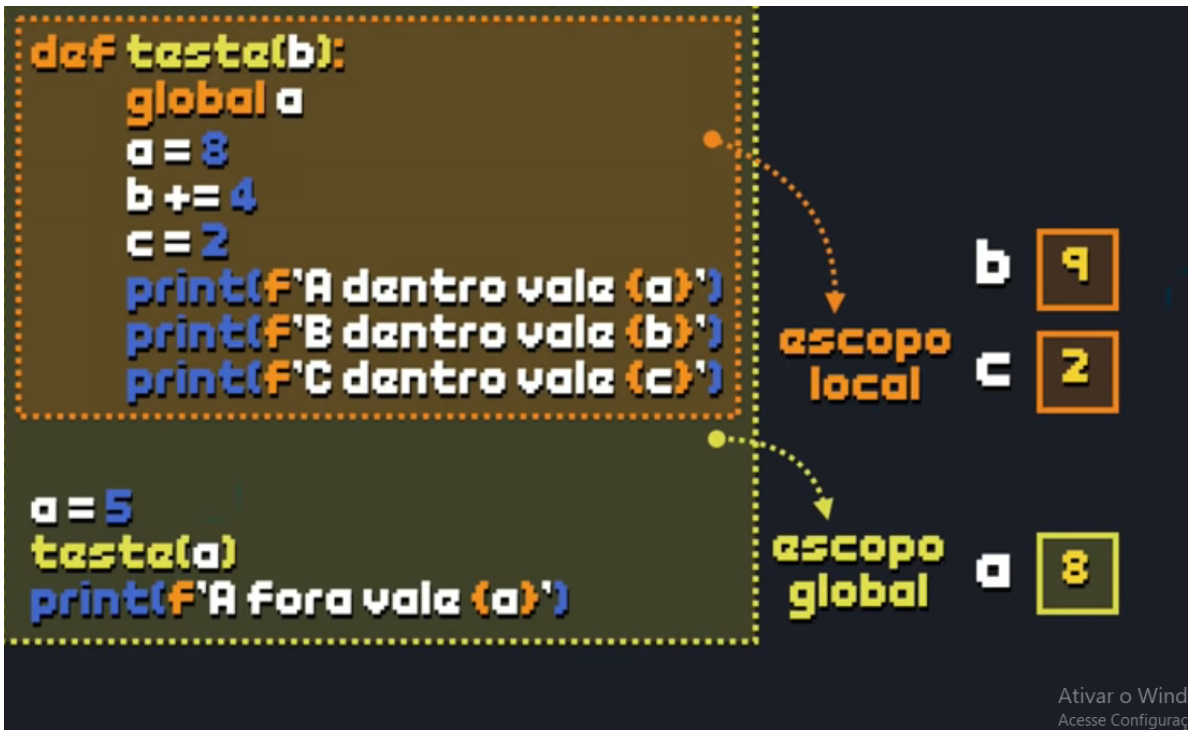


Dentro da função eu posso utilizar a variável global escrevendo

global nome_variável

dessa forma mesmo que exista uma **variável_local** com o mesmo nome da

variável_global, o python vai priorirar / decidir que o valor usado é da **variável_global**

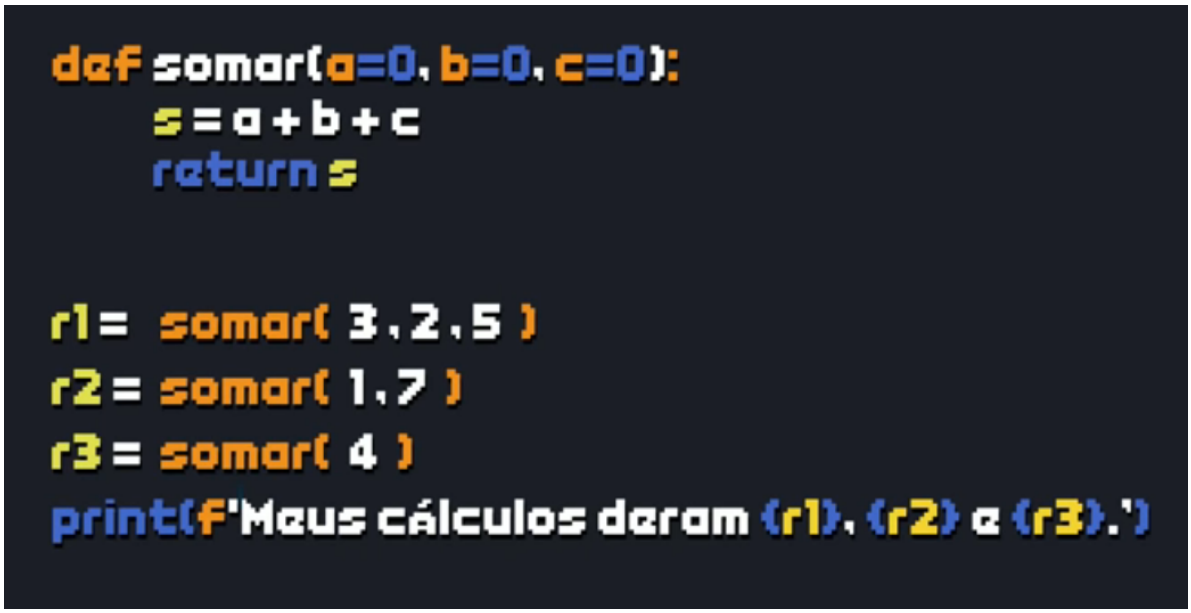


dentro do código não vai ser criado uma nova **variável_local** a

porque eu quero usar a **variável_global** que é a prioridade e depois atribui o valor 8 para a **variável_global.** O valor atribuido a variável_global obviamente vai valer esse valor dentro e fora da função

Retorno de valores

Exemplos de retorno.



Compreensão de Lista

List comprehension

É uma forma simples e concisa de criar uma lista. Podemos aplicar condicionais e laços para criar diversos tipos de listas a partir de padrões que desejamos para a nossa estrutura de dados.

Arquivo com exemplos da **List comprehension**

Cópia_de_List_Comprehensions.ipynb

```
# Código que eu criei para criar uma lista apenas com os números pares

[ i for i in range(21) if i % 2 == 0]

Resultado
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

A gente só precisou de uma única linha para criar isso, economizando até
mesmo poder de processamento e agilizando a execução
```

Aula Alura

Situação 9:

Agora, precisamos utilizar as médias calculadas no exemplo anterior, pareando com o nome dos estudantes. Isto será necessário para gerar uma lista que selecione aqueles estudantes que possuam uma média final maior ou igual a 8 para concorrer a uma bolsa para o próximo ano letivo. Os dados recebidos correspondem a uma lista de tuplas com os nomes e códigos dos estudantes e a lista de médias calculadas logo acima.

Vamos resolver esse desafio?

Para facilitar o nosso entendimento do processo vamos trabalhar com uma turma fictícia de 5 estudantes.

Dica: Utilize o formato:

```
[expr for item in lista if cond]
```

```
nomes = [('João', 'J720'), ('Maria', 'M205'), ('José', 'J371'), ('Cláudia', 'C546'), ('Ana', 'A347')]
medias = [9.0, 7.3, 5.8, 6.7, 8.5]
```

```
# Gerando a lista de nomes (extraindo da tupla)
nomes = [nome[0] for nome in nomes]
nomes

['João', 'Maria', 'José', 'Cláudia', 'Ana']
```

Dica: Para conseguirmos parear as médias e nomes facilmente, podemos recorrer a mais uma built-in function: `zip()`

Ela recebe um ou mais iteráveis (lista, string, dict, etc.) e retorna-os como um iterador de tuplas onde cada elemento dos iteráveis são pareados.

```
estudantes = zip(nomes, medias)
estudantes

<zip at 0x7ea9b353a500>

#Gera um objeto do tipo zip
#Agora eu preciso transformar esse código zip em uma lista para conseguir ler
#Para isso tenho que fazer isso - colocar a função list()

estudantes = list(zip(nomes, medias))
estudantes

[('João', 9.0), ('Maria', 7.3), ('José', 5.8), ('Cláudia', 6.7), ('Ana', 8.5)]


 estudantes que possuam uma média final maior ou igual a 8 para concorrer a uma
 bolsa para o próximo ano letivo.
# Gerando a lista de pessoas candidatas a bolsa
candidatos = [ selecionado for selecionado in estudantes if selecionado[1] >= 8]
candidatos
```

Para saber mais: função zip

Para saber mais: função zip

Dict comprehension

É uma forma simples e concisa de criar ou modificar um dicionário. Podemos aplicar condicionais e laços para criar diversos tipos de dicionários a partir de padrões que desejamos para a nossa estrutura de dados e com o suporte de iteráveis como listas ou sets.

https://peps.python.org/pep-0274/

Selecionando bolsistas

Recebemos uma demanda da instituição de ensino do nosso projeto. Foi repassado para nós uma lista de 20 estudantes e suas respectivas médias finais. Aqui, nós precisamos selecionar estudantes que tenham média final maior ou igual a 9.0. Esses(as) estudantes serão premiados(as) com uma bolsa de estudos para o próximo ano letivo.

Para filtrar os dados, temos que gerar um dicionário cujas chaves são os nomes e os valores são as médias dos(as) estudantes selecionados(as). Estes são os dados recebidos:

```
nomes_estudantes = [ "Enrico Monteiro", "Luna Pereira", "Anthony Silveira", "Letícia Fernandes",
                     "João Vitor Nascimento", "Maysa Caldeira", "Diana Carvalho", "Mariane da Rosa",
                     "Camila Fernandes", "Levi Alves", "Nicolas da Rocha", "Amanda Novaes",
                     "Laís Moraes", "Letícia Oliveira", "Lucca Novaes", "Lara Cunha",
                     "Beatriz Martins", "João Vitor Azevedo", "Stephany Rosa", "Gustavo Henrique Lima" ]

medias_estudantes = [5.4, 4.1, 9.1, 5.3, 6.9, 3.1, 10.0, 5.0, 8.2, 5.5,
                     8.1, 7.4, 5.0, 3.7, 8.1, 6.2, 6.1, 5.6, 6.7, 8.2]
```

Qual dos códigos abaixo representa a forma correta de gerar o dicionário com os(as) estudantes selecionados(as)?

> Dica ⇒ utilize a forma:
>
> ```
> {expressao_chave: expressao_valor for item in iteravel if con
> ```
>
> ```
> bolsistas = {nome: media
>              for nome, media in zip(nomes_estudantes, medias_estudantes)
>              if media >= 9.0}
> bolsistas
> ```

Essa é uma das formas para gerar o dicionário dos(as) estudantes com média acima de 9.0 levando em consideração as listas passadas. O zip aqui auxilia na construção dos pares de nome e média de cada estudante e cada par é iterado e filtrado pela condição para a construção da chave e valor do dicionário.

Errado

```
bolsistas = {nomes_estudantes: medias_estudantes
             for i in range(len(medias_estudantes))
             if medias_estudantes[i] >= 9.0}
bolsistas
Saída: {'Anthony Silveira': 9.1, 'Diana Carvalho': 10.0}
```

Neste exemplo, lemos a posição dos elementos do iterável da média. Mas passamos para chave e valor a lista inteira dos nomes e média de uma vez, o que gerará um erro.

Desafio hora pratica

Para saber mais: tipos de exceções