Tipos Básicos e Variáveis

Variável é um recurso primal de qualquer linguagem de programação para armazenar valores em memória. Ao declarar uma variável, podemos gravar e ler esses valores diretamente dela a partir do nome que definimos para a variável.

Atribuir variáveis em Python é fácil: basta colocar um nome de variável à esquerda de um sinal de igualdade =:

# a variável x recebe o valor 4

x = 4

Isso pode parecer óbvio, mas geralmente quando estamos lendo algo no cotidiano, o sinal de = está associado a uma comparação, já no Python este operador é associado com **atribuição**. Se você tiver o modelo mental errado na hora de interpretar o que esta operação faz, a forma como o Python funciona pode parecer confusa.

Nomenclatura das Variáveis

Antes que você possa escrever sem muito esforço um código bonito e livre de bugs, você deve entender porquê ele é bonito. Os códigos são lidos muito mais frequentemente do que são escritos, então a legibilidade faz toda a diferença.

* Os nomes das variáveis têm comprimento ilimitado
* Os nomes das variáveis devem começar com uma letra ou um sublinhado \_, seguido de letras, números ou sublinhados.
* Os nomes das variáveis são sensíveis a letras maiúsculas/minúsculas
* Os nomes das variáveis não podem ser palavras-chave

Os nomes variáveis convencionalmente devem ter letras minúsculas, com múltiplas palavras separadas por sublinhados.

# Nome da variável | Valor atribuido a variável

minha\_variavel = 'meu valor em texto'

**Tipos primitivos**

Existem alguns tipos básicos de variável em Python que nos ajudam a ter mais controle sobre a lógica na hora desenvolvimento, algumas delas são:

Tipos básicos:

* int

De **integer,** denota que o valor da variável é um algarismo numérico **inteiro**.

* float

Denota que o valor da variável é um número '**flutuante**' ou **real**, com **casas decimais**.

Leia: [Números em Python](https://docs.python.org/pt-br/3/tutorial/introduction.html#numbers)

* **str**

 De **string**, denota que o valor da variável é um ou mais **caracteres**.

* **bool**

Existem dois tipos incorporados que representam os estados binários **True** e **False**. Um objeto booleano pode ser usado para registrar o estado de algo, por exemplo, se uma tarefa foi concluída ou não, ou se um dispositivo está ligado ou não.

**Leia:**[**Valores Booleanos**](https://docs.python.org/pt-br/3/library/stdtypes.html#boolean-values)

x = 1 # x é um inteiro

x = 'Olá' # agora x é uma string

x = 1.5 # agora x é um float

x = [1, 2, 3] # agora x é uma lista de inteiros

x = (1, 2, 3,) # agora x é uma tupla de inteiros

x = True # agora x é um boleano de valor True

x = {'chave\_1': 42, 'chave\_2': 500} # agora x é um dicionário

Agrupadores de dados:

* **list**

Utilizada para **agrupar valores**, a qual cada elemento da lista pode ser acessado por um **indice numérico inteiro**, sendo o primeiro elemento o de índice **0**, o segundo de índice **1** e assim por diante. Uma lista pode conter vários dos tipos de variáveis citadas anteriormente, e pode ser alterada conforme o necessário.

x = [1 , 2, 3] # x é uma lista de inteiros

x = ['olá', '0', 'ao', 'python'] # x agora é uma lista de strings

x = ['1', 'sim', 2, 'olá', 5] # x agora é uma lista que contém tanto inteiros quanto strings

# Acessando determinado elemento

x = ['Zero', 'ao', 'Python']

x[0]

> 'Zero'

x[1]

> 'ao'

x[2]

> 'Python'

# Reatribuindo valores a um indice de uma lista já criada

x = ['Zero', 'ao', 'Python']

x[0] = 'Zerooooo'

x

> ['Zerooooo', 'ao', 'Python']

* **tuple**

Utilizada para **agrupar valores**,sua diferença para a lista é que a tupla **não pode ser modificada** uma vez criada. Também é possivel acessar os elementos da tupla a partir de um indice numérico inteiro.

# Declaração de uma tupla

minha\_tupla = ('Zero', 'ao', 'Python',)

# Acessando determinado elemento

minha\_tupla[0]

> 'Zero'

minha\_tupla[2]

> 'Python'

# Tentando reatribuir um valor a um indice da tupla ja criada (error)

minha\_tupla[0] = 'Zerooooooou'

> TypeError: 'tuple' object does not support item assignment

* **dict**

De **dictionary**, utilizado para **agrupar valores** com **chaves únicas**. Ao contrário das listas e das tuplas, que só suportam indices numéricos inteiros começando em 0, o dicionário nos permite customizar nossos índices para comportar qualquer tipo basico. A esses índices customizáveis damos o nome de **key**ou**chave**, e ao dado associado a ele pelos dois pontos damos o nome de **value** ou **valor**.

         Observe que no exemplo, a primeira chave 'chave\_1' é do tipo **string** e seu valor é 20 do tipo int. Ja a segunda chave 2 é do tipo **int**e seu valor 'valor\_2' é do tipo **string**

# Declarando um dicionário

meu\_dict = {'chave\_1': 20, 2: 'valor\_2'}

# Acessando um determinado valor de um dicionário

my\_dict['chave\_1']

> 20

1. Variable assignment

Arrume as declarações de variáveis de uma forma que o código guarde a string 'devLab' na variável name

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/5612e743cab69fec6d000077/train/python)

2. Grasshopper - Messi Goals

Utilize variáveis para descobrir a soma dos gols que o Messi fez em 3 competições.

**Informações**

|  |  |
| --- | --- |
| **Competição** | **Gols** |
| La Liga | 43 |
| Champions League | 10 |
| Copa del Rey | 5 |

**Tarefa**

**1.** Criar as variáveis e guardar os valores apropriados utilizando a tabela acima:

* la\_liga\_goals
* champions\_league\_goals
* copa\_del\_rey\_goals

**2.** Criar uma quarta variável chamada total\_goals que guardará a soma de todos os gols do Messi neste ano.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/55ca77fa094a2af31f00002a/train/python)

3. Opposite Number

Muito simples, dado um número, encontre o seu oposto.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/56dec885c54a926dcd001095/train/python)

**Exemplo**

opposite(1) # Retorno -> -1

opposite(-45) # Retorno -> 45

4. You Can't Code Under Pressure #1

Code o mais rápido que conseguir! Você precisa dobrar o inteiro e retorna-lo.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/53ee5429ba190077850011d4/train/python)

**Exemplo**

double\_integer(5) # Retorno -> 10

double\_integer(10) # Retorno -> 20

# Estruturas de decisão

Temos como algumas estruturas de decisão como if, elif e else. Se (if) uma determinada condição for atendida, faça algo — senão, se (elif) outra condição for atendida, faça algo diferente — senão, (else) faça algo quando nenhuma das condições forem atendidas. Cada condição termina com dois pontos : para indicar que um novo bloco de código indentado vem depois dele. O bloco de código identado após cada estrutura condicional, será executado caso a condição seja atendida.

**if ->** executado apenas quando a condição for verdadeira.

if 1 == 1:

print('passsou no if')

> passou no if

**if... elif ->** executa apenas a condição que for verdadeira.

if 1 == 2:

print('passsou no if')

elif 1 == 1:

print('passou no elif')

> passou no elif

**if... else ->** executa o if se a condição for verdadeira, senão passa para o else.

if 1 == 2:

print('passsou no if')

else:

print('passou no else')

> passou no else

1. Drink about

* Sua função irá receber a idade (integer) e deve retornar a bebida de acordo com a idade:
  + Se a idade for **menor que 14** deve retornar 'drink toddy';
  + Se a idade for **menor que 18** deve retornar 'drink coke';
  + Se a idade for **menor que 21** deve retornar 'drink beer';
  + Se a idade for **maior ou igual a 21** deve retornar 'drink whisky';

**Para realizar essa prática utilize este**[**Link do Codewars**](https://www.codewars.com/kata/56170e844da7c6f647000063/train/python).

**Exemplo**

people\_with\_age\_drink(13) # Retorno -> "drink toddy"

people\_with\_age\_drink(17) # Retorno -> "drink coke"

people\_with\_age\_drink(18) # Retorno -> "drink beer"

people\_with\_age\_drink(20) # Retorno -> "drink beer"

people\_with\_age\_drink(30) # Retorno -> "drink whisky"

2. Is he gonna survive?

Ele irá sobreviver? Um herói está a caminho do castelo para completar sua missão. Entretanto, lhe foi dito que o castelo é protegido por alguns dragões muito poderosos! Cada dragão necessita de duas munições para ser derrotado, e o nosso herói não faz ideia de quantas munições ele deve levar... Assumindo que ele irá pegar um número específico de munições e partirá para a batalha contra um número específico de dragões, ele irá sobreviver?

**Nota:** Você receberá a quantidade de balas do herói e a quantidade de dragões do castelo.

**Para realizar essa prática utilize este**[**Link do Codewar**](https://www.codewars.com/kata/59ca8246d751df55cc00014c/train/python).

**Exemplo**

hero(10, 5) # Retono -> True

hero(15, 9) # Retono -> False

hero(100, 50) # Retono -> True

hero(1050, 751) # Retono -> False

hero(2, 1) # Retono -> True

hero(1, 1) # Retono -> False

 Olá, desenvolvedores! O nosso conteúdo do dia englobará algumas das estruturas fundamentais do Python, as strings. Esperamos que vocês possam tirar o máximo possível de proveito do conteúdo que estamos disponibilizando, não esqueçam de praticar bastante para fixar bem.

Strings

Um objeto string em Python tem informações de caracteres e pode conter qualquer número de caracteres, inclusive zero. Muitas vezes você verá esse tipo de dado abreviado como *str*.

As strings são indicadas usando aspas simples, aspas duplas ou aspas triplas. As aspas simples são utilizadas para maioria dos casos. Se quisermos inserir aspas simples dentro da string, teremos duas alternativas, a primeira alternativa é usar o operador de escape \ antes das aspas, e a segunda alternativa é envolver sua string com aspas duplas. O mesmo vale para strings que contém aspas duplas, precisamos ou anular esse caractere com o operador de escape \ ou envolver a string com aspas simples. As aspas triplas são usadas para strings longas que podem se estender por várias linhas. O tipo de aspa usada no final da string deve corresponder ao tipo de aspa do começo.

As aspas simples podem conter aspas duplas dentro delas, e vice-versa. Aqui vai alguns exemplos.

my\_name = 'Kenzie' # single-quoted string

mixed\_quote = "Kenzie won't let me down" # double-quoted string with single embedded quote

multi\_line = """Red sky at night:

sailor's delight. Red sky at morning:

sailor's warning."""

escaped\_string = 'Here\'s looking at you, kid'

empty\_string = ''

Indexação e Fatiamento

As strings são consideradas *listas* em Python, e, portanto, os caracteres dentro delas podem ser acessados via índice ou *fatiamento*.

Índices:

Para acessar um caractere em uma posição específica dentro de uma string, coloque um inteiro que representa a posição do caractere entre colchetes, [].

Os números positivos começam pela esquerda, pelo 0, e os números negativos começam pela direita, pelo -1.

s = "some string"

s[0] # s

s[3] # e

s[-1] # g

Fatiamento

Para acessar grupos de caracteres dentro de uma string, chamados *fatias*, use dois pontos (:) para separar a posição inicial da posição final. A sintaxe se parece com isso string[start:stop:step], onde *start*, *stop* e *step* são inteiros.

* A posição *start* está inclusa na fatia resultante.
* A posição *stop* **não** está inclusa na fatia resultante.
* O valor de *step* indica quantos caracteres devem ser pulados por vez.
* Todos os valores são opcionais, desde que seja incluído um dois pontos para representar a operação de fatiamento.

s = "some string"

s[0:4] # valores de start e stop explícitos: 'some'

s[5:11:2] # inclui um valor step -- fatia a cada duas posições da posição 5 até a posição 11: 'srn'

s[:4] # omite o valor para start -- o valor 0 fica implícito por default: 'some'

s[5:] # omite o valor para stop -- a última posição da string fica implícita por default 'string'

s[:] # omitir o valor para start, stop e step 'some string'

Você pode usar até mesmo valores negativos para criar fatias interessantes.

s = "some string"

s[-6:] # inicia 6 caracteres antes do caractere final: 'string'

s[:-7] # para o fatiamento 7 caracteres antes do caractere final:'some'

s[::-1] # inverte a string: 'gnirts emos'

Métodos de Strings

Vamos explorar alguns métodos incorporados que vem com cada objeto string.

capitalize() - muda o primeiro caractere da string para maiúsculo

>>> s = "some string"

>>> s.capitalize()

'Some string'

count() - retorna o número de ocorrências de uma determinada sub-string da string

>>> s = "some string for me"

>>> s.count("me")

2

endswith() - informa se a string termina com uma determinada sub-string

>>> s = "some string"

>>> s.endswith("ing")

True

find() - retorna o primeiro índice de onde uma determinada sub-string é encontrada dentro da string (retorna -1 se não encontrar nada)

>>> s = "some string"

>>> s.find("me")

2

>>> s.find("x")

-1

index() - o mesmo que find(), mas retorna uma exceção ValueError caso a sub-string não seja encontrada

isalpha(), isdigit(), isspace () - informa se a string satisfaz os critérios de alfanumérico, número e espaço

Há um monte desses tipos de métodos que verificam as características de uma string. Digite dir(str) no REPL para consultá-los.

>>> s = "some string"

>>> s.isalpha() # retorna False por conta do espaço

False

join() - insere a string entre cada item de um determinado dado iterável

Este método é frequentemente usado anexado diretamente a uma string literal para ser usado como um separador para outros dados.

>>> s = "xyz"

>>> ".".join(s) # put dots in between each letter of s

'x.y.z'

lower() - retorna uma versão em letras minúsculas da string

>>> s = "SOME STRING"

>>> s.lower()

'some string'

replace() - substitui todas as ocorrências de uma sub-string da string por uma determinada string

>>> s = "some string"

>>> s.replace("s", "z")

'zome ztring'

split() - retorna uma lista de palavras de uma string, com um separador opcional para ser usado como delimitador

>>> s = "some string"

>>> s.split() # default usage

['some', 'string']

>>> s.split("t")

['some s', 'ring']

startswith() - informa se uma string começa com uma determinada sub-string

>>> s = "some string"

>>> s.startswith("so")

True

strip() - remove os espaços em branco iniciais e finais do texto

Opcionalmente, você pode indicar um caractere alternativo para remover do início e do final do texto.

>>> s = " some string "

>>> s.strip()

'some string'

>>> s = "some strings"

>>> s.strip("s")

'ome string'

upper() - retorna uma versão maiúscula da string

>>> s = "some string"

>>> s.upper()

'SOME STRING'

Esperamos que você tenha aproveitado ao máximo o nosso conteúdo de loops. Conte conosco para o que precisar!

1. Reversed Words

Complete a solução de modo que ela inverta todas as palavras da string passada por parâmetro

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/51c8991dee245d7ddf00000e/train/python)

**Exemplo**

reverseWords('A maior vitória é a que não requer nenhuma batalha')

# Deverá retornar:

> 'batalha nenhuma requer não que a é vitória maior A'

2. Short Long Short

* Dadas 2 strings, a e b
  + Retorne uma string no formato **menor+maior+menor**, com a **string menor por fora** e a **string maior por dentro**.
* As strings passadas nunca serão de tamanho igual, mas elas podem ser vazias (length 0).

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/50654ddff44f800200000007/train/python)

**Exemplo**

solution('1', '22') # Retorno -> '1221'

solution('22', '1') # Retorno -> '1221'

solution('', 'Kenzie') # Retorno -> 'Kenzie'

solution('Python', '') # Retorno -> 'Python'

3. Returning Strings

Faça uma função que irá retornar uma mensagem de boas vindas que utilize uma entrada name.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/55a70521798b14d4750000a4/train/python)

**Exemplo**

greet('Luan') # Retorno -> 'Hello, Luan how are you doing today?'

greet('Guilherme') # Retorno -> 'Hello, Guilherme how are you doing today?'

Listas e métodos de lista

As listas são um dos tipos de dados de sequência do Python e são úteis para armazenar dados relacionados em grupos usando apenas um nome. Elas são muito versáteis, permitindo armazenar dados de qualquer tipo dentro delas. Você irá usá-las o tempo todo. Aqui estão algumas características das listas.

* Elas são mutáveis - podem ser modificadas diretamente após a criação sem criar um novo objeto;
* São sequenciais - seus conteúdos são ordenados e podem ser acessados através de sua posição dentro da sequência.

Criando uma Lista

Como é o caso de alguns outros tipos de dados, há mais de uma maneira de criar uma lista: a sintaxe literal [] e a função incorporada list().

>>> ["a", "b", "c"]

['a', 'b', 'c']

>>> list("abc")

['a', 'b', 'c']

As duas linhas de código acima resultam na mesma lista. A diferença é que a função list() leva um iterável(um objeto em que pode ser feito um loop sobre) como argumento e popula a lista resultante com cada um dos valores do iterável. No exemplo acima, "abc" é uma string, que é iterável, e retorna um caractere por vez quando é iterada.

Acessando os Itens de uma Lista

Assim como as strings, as listas podem ser indexadas e fatiadas. Isso possível devido à sua característica sequencial.

>>> my\_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> my\_list[0]

1

>>> my\_list[-2]

9

>>> my\_list[5:]

[6, 7, 8, 9, 10]

Removendo os Itens de uma Lista

Isso mesmo! Há várias maneiras de fazer isso. Você será apresentado a alguns métodos na seção seguinte, mas aqui estão algumas maneiras de fazer isso com palavras-chave Python e fatiamento.

A remoção de fatias de uma lista permite que você remova pedaços de uma só vez. Basta atribuir um iterável falso (e.g., "", (), [], {}) e toda a fatia é então removida da lista. É comum usar um literal de string vazia, "". Lembre-se, o fim da fatia não é incluído na própria fatia (ou seja, my\_list[2:5] inclui os índices 2, 3 e 4, mas não o índice 5).

Métodos da Lista

Os métodos de lista incorporados permitem que você efetue operações em uma lista. Observe como alguns deles modificam a lista localmente, sem retornar uma nova lista. Isso se deve à característica da lista de ser mutável. Às vezes, isso pode ser uma surpresa se você estiver esperando uma lista como um valor de retorno, já que algumas delas retornam um tipo Nenhum.

append() - adiciona um item ao fim de uma lista

>>> my\_list = [1, 2]

>>> my\_list.append(3)

>>> my\_list

[1, 2, 3]

clear() - remove todos os itens de uma lista

>>> my\_list = [1, 2]

>>> my\_list

[1, 2]

>>> my\_list.clear()

>>> my\_list

[]

Criando cópias de listas

copy() - retorna uma cópia de uma lista

>>> my\_list1 = [1, 2, 3]

>>> my\_list2 = my\_list1.copy()

>>> my\_list2

[1, 2, 3]

>>> id(my\_list1), id(my\_list2) # not the same object

(4305012544, 4305012608)

É importante observe que isso produz uma cópia rasa de my\_list1. Isso afetará a forma como as alterações dos itens da lista são tratadas.

count() - conta o número de ocorrências de um valor a partir de uma lista

>>> my\_list = [1, 2, 3, 2, 1]

>>> my\_list.count(3)

1

>>> my\_list.count(2)

2

extend() - adiciona itens de um iterável ao final de uma lista

>>> my\_list = [1, 2, 3]

>>> my\_list.extend([4, 5, 6])

>>> my\_list

[1, 2, 3, 4, 5, 6]

* Pode ser fácil confundir extend() com append(), uma vez que ambos adicionam algo ao final de uma lista. Tente lembrar-se que o append() adiciona o objeto inteiro à lista…
* >>> my\_list = [1]
* >>> my\_list.append([2, 3])
* >>> my\_list

[1, [2, 3]]

…mas o extend() adiciona cada item de um iterável.

* >>> my\_list = [1]
* >>> my\_list.extend([2, 3])
* >>> my\_list
* [1, 2, 3]

index() - obtém o índice da primeira ocorrência de um valor em uma lista

>>> my\_list = [1, 2, 3, 2, 1]

>>> my\_list.index(2)

1

Veja você mesmo o que acontece quando o valor que você está procurando não está na lista. Mais adiante nesse curso, discutiremos como lidar apropriadamente com as exceções.

insert() - adiciona um item em qualquer posição de uma lista

>>> my\_list = [1, 3]

>>> my\_list.insert(1, 2)

>>> my\_list

[1, 2, 3]

pop() - remove o último(se não for passado parâmetro) e/ou retorna um item de qualquer posição de uma lista(caso o parâmetro seja informado)

>>> my\_list = [1, 2, 3]

>>> my\_list.pop()

3

>>> my\_list.pop(0)

1

>>> my\_list

[2]

Observe que, sem argumentos, o pop() remove e retorna o último item da lista. A palavra "pop" é parte da terminologia utilizada ao se discutir estruturas de dados como pilhas, o que veremos em breve.

remove() - remove a primeira ocorrência de um valor de uma lista

>>> my\_list = [1, 3, 2, 3]

>>> my\_list.remove(3)

>>> my\_list

[1, 2, 3]

reverse() - inverte a ordem dos itens de uma lista

>>> my\_list = [1, 2, 3]

>>> my\_list.reverse()

>>> my\_list

[3, 2, 1]

Ordenando Listas

sorted() - ordena os itens de um iterável e os retorna em forma de lista

list.sort() - ordena os itens de uma lista in loco

Os itens da lista podem ser ordenados usando a função incorporada do Python sorted(), ou o método sort() de um objeto de lista. A diferença é que sort() é um método em um objeto de lista que ordena a lista in loco…

>>> my\_list = [3, 2, 1]

>>> my\_list.sort()

>>> my\_list

[1, 2, 3]

…já a função incorporada sorted() pega um iterável e retorna uma lista de seus itens ordenados.

>>> my\_list = [3, 2, 1]

>>> my\_list\_sorted = sorted(my\_list)

>>> my\_list\_sorted

[1, 2, 3]

Ambos permitem algum tipo de personalização na forma como a lista é ordenada ao usar um par de argumentos de palavra-chave.

key - uma função que será aplicada a cada item, e terá seu resultado usado como o valor de ordenação

>>> my\_list = ["hello", "hi", "greetings", "hey"]

>>> my\_list.sort(key=len)

>>> my\_list

['hi', 'hey', 'hello', 'greetings']

reverse - um indicador booleano para indicar a direção da ordenação

>>> my\_list = [1, 2, 3]

>>> my\_list.sort(reverse=True)

>>> my\_list

[3, 2, 1]

# 1. Find the smallest integer in the array

Dado uma lista de inteiros, sua solução deverá retornar o menor número da lista.

Você pode assumir, para o proposito dessa prática, que nunca passaremos uma lista vazia para a sua função.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/55a2d7ebe362935a210000b2/train/python)

**Exemplo**

find\_smallest\_int([34, 15, 88, 2]) # Retorno -> 2

find\_smallest\_int([34, -345, -1, 100]) # Retorno -> -345

# 2. Grasshopper - Array Mean

Ache a média de uma lista de números.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/55d277882e139d0b6000005d/train/python)

**Informação**

Para achar a média de um conjunto de números, some todos eles e divida pela quantidade de números na lista.

Por exemplo, considerando uma lista com os seguintes números [1, 3, 5, 7]

1. Some todos os números da lista:

1 + 3 + 5 + 7 = 16

2. Divida esse numero ( 16 ) pela quantidade de números da lista. Neste exemplo temos 4 números na lista.

16 / 4 = 4

 3. A média dessa lista é 4

# Tuplas

As tuplas, assim como as listas, são um dos tipos básicos de dados de sequência do Python. Entretanto, elas se diferem das listas no sentido em que são imutáveis. Elas são o tipo de dado perfeito para quando você precisa armazenar dados agrupados de forma que eles não possam ser modificados, tais como as constantes para usar ao longo de um programa. Nesta aula, destacaremos as características e métodos das tuplas.

## Sintaxe das tuplas

>>> animals = ("dog", "cat", "squirrel") # a tuple created using parentheses

>>> animals

('dog', 'cat', 'squirrel')

>>> type(animals)

<class 'tuple'>

>>> animals = "dog", "cat", "squirrel" # a tuple created without using parentheses

>>> animals

('dog', 'cat', 'squirrel')

>>> type(animals)

<class 'tuple'>

Uma tupla de elemento único pode ser criada através da colocação de uma vírgula depois de um único valor, entre parênteses ou não.

>>> favorite\_animal = ("sloth",) # perceba a virgula colocada após as aspas duplas

>>> favorite\_animal

('sloth',)

>>> type(favorite\_animal)

<class 'tuple'>

>>> favorite\_animal = "sloth", # perceba a vírgula colocada após as aspas duplas

>>> favorite\_animal

('sloth',)

>>> type(favorite\_animal)

<class 'tuple'>

## Acessando Tuplas

Os elementos de uma tupla podem ser acessados da mesma forma que acessamos as listas, com indexação e fatiamento.

>>> animals = ("dog", "cat", "squirrel")

>>> animals[0]

'dog'

>>> animals[1:]

('cat', 'squirrel')

## As Tuplas são Imutáveis

Os valores de uma tupla não podem ser alterados. Se você tentar reatribuir um índice de uma tupla, uma exceção será exibida.

>>> animals = ("dog", "cat", "squirrel")

>>> animals[2] = "moose"

Traceback (most recent call last):

File "", line 1, in

TypeError: 'tuple' object does not support item assignment

Observação: estruturas mutáveis aninhadas dentro de uma tupla, no entanto, podem ser modificadas (por exemplo, uma lista dentro de uma tupla), o que pode resultar em algumas coisas diferentes que você deve ficar atento quando começar a usar o Python de formas mais avançadas.

## Desempenho das Tuplas

A imutabilidade das tuplas é importante para entender seu uso. Considere adicionar itens a uma lista. Como uma lista é mutável, ela pode ser modificada sem criar qualquer novo objeto na lista. Aqui está um exemplo de atribuição de 10 número a uma lista, um de cada vez.

>>> my\_list = []

>>> for i in range(10):

... my\_list.append(i) # modify my\_list in place

...

>>> my\_list

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Agora, considere tentar o mesmo tipo de operação com uma tupla. Devido a sua imutabilidade, um novo objeto tupla deve ser criado em cada iteração.

>>> my\_tuple = ()

>>> for i in range(10):

... my\_tuple += (i,) # create a new object each time

...

>>> my\_tuple

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

A mutabilidade é importante para o desempenho quando você sabe que estará modificando frequentemente um objeto. Neste caso, use uma lista para evitar a criação de um novo objeto cada vez que ela for modificada. As tuplas, por outro lado, são a escolha perfeita para os dados que você sabe que não serão modificados.

## Métodos de Tuplas

count() - retorna o número de ocorrências de um determinado valor na tupla

>>> nums = (11, 33, 42, 50, 33, 98)

>>> nums.count(33)

2

index() - retorna o índice da primeira ocorrência de um determinado valor na tupla

>>> nums = (11, 33, 42, 50, 33, 98)

>>> nums.index(33)

1

E esses são todos os métodos das tuplas! O poder delas está em sua implementação e utilização, em vez de numa abundância de métodos de manipulação de dados, uma vez que,

# Prática de Tuplas: criando e concatenando tuplas

Olá, desenvolvedores. Nessa prática, vocês serão desafiados a criarem duas tuplas e juntar as duas em uma terceira tupla. Exemplo:

a = (1, 2, 3, 4,)

b = ('a', 3, 1, 't',)

c = ...

print(c)

> (1, 2, 3, 4, 'a', 3, 1, 't')

Esperamos que você aproveite ao máximo a prática, pode contar conosco para qualquer dúvida!

Loops

Olá, desenvolvedores! O nosso conteúdo do dia englobará algumas das estruturas fundamentais do Python, os nossos loops. Esperamos que vocês possam tirar o máximo possível de proveito do conteúdo que estamos disponibilizando, não esqueçam de praticar bastante para fixar bem.

for…in

O Python não remove e tem loops *for* no sentido tradicional. Ele tem apenas loops *for...in*, o qual chamaremos de loop *for*. O loop *for* do Python pega um iterável e retorna cada item dele, um por vez, até que todos os itens tenham sido iterados ou o loop tenha terminado. Ele funciona no estilo *iteração* e *iterável*, que veremos mais detalhadamente numa aula futura.

No exemplo a seguir, carrinho\_compras  é uma lista de itens de compra . Dentro do loop *for*, você tem acesso à iteração atual do item a cada passagem pelo loop.

for item in carrinho\_compras:

print(item)

Aqui está um exemplo da impressão de cada letra de uma string, uma de cada vez.

for letra in "Olá Mundo":

print(letra)

Agora, podemos subir o nível do código Yoda para que o usuário tenha um certo número de tentativas.

for \_ in range(3):

print("Options:\n1 - Do\n2 - Do Not\n3 - Try")

choice = int(input("Choose one: "))

if choice == 1:

print("Yes. Good.\n")

break

elif choice == 2:

print("That is why you fail.\n")

elif choice == 3:

print("Do. Or do not. There is no Try.\n")

else:

print("You seek Yoda.\n")

Essas mudanças introduzem duas novas ideias.

O break - sai do loop e continua a execução do programa no próximo statement

Assim que o usuário digitar 1, precisamos terminar o loop, então usamos um statement break.

* + O statement break tem um primo: continue. O uso de um statement continue dentro de um loop faz com que ele comece a próxima iteração imediatamente, saltando qualquer trecho de código que restar na iteração atual.

O  **\_**  - o caractere sublinhado tem muitos usos diferentes em Python, mas neste contexto é usado como uma variável descartável, uma vez que não planejamos usar os valores que vierem de range().

* Se quisermos usar os valores que range() produz a cada iteração, é comum armazená-los sob o nome  i , como em for i in range(3):.

|  |
| --- |
| Melhores Amigos para Sempre  Diga "oi" aos melhores amigos do loop *for*!    enumerate() - obtém tanto um índice quanto um valor a partir de um iterável  Esta função incorporada pareia cada item de um iterável com a sua posição dentro do iterável. Isso é extremamente comum, e é especialmente útil quando você precisa verificar outras posições dentro do iterável a partir do contexto do item atual. Considere um exemplo onde queremos fazer um loop em uma lista de números e descobrir se a soma de quaisquer dois números de uma linha é igual a 10.  found = False  nums = [3, 4, 7, 2, 8, 4, 6]  for i, num in enumerate(nums):  if i > 0 and num + nums[i-1] == 10:  found = True  print(found)    range() - gera uma sequência de inteiros com base nos valores indicados  Esta função incorporada pode ser usada para emular o tradicional loop *for* ao fazer o loop um determinado número de vezes. Ele tem duas formas principais. Se você fornecê-lo com apenas um argumento, ele vai começar em 0 e usará o seu argumento como o valor *stop* (sem incluí-lo).  for i in range(10):  print(i)  O código acima imprimirá 0-9. É muito comum cometer o erro de pensar que o intervalo inclui o valor do *stop*. Neste caso, use range(11) para incluir o número 10.  A outra forma consiste em dois ou três argumentos. Ao usar mais de um argumento, o primeiro argumento se torna o valor *start*, o segundo é o *stop*, e o terceiro argumento é opcional, e é o valor *step*, ou incremento. Então, se você quisesse começar em 1 no exemplo acima, você poderia fazer o seguinte.  for i in range(1, 11):  print(i)  E, por fim, o terceiro valor que podemos passar como parâmetro para range é o "step", isto é, de quantos em quantos elementos acontecerá a iteração, colocando o valor 3 como step por exemplo, teremos o i iterando a cada três elementos.  for i in range(1, 11, 3):  print(i) |

while

O loop *while* é usado quando você quer continuar fazendo algo enquanto uma condição é verdadeira ou até que o loop seja encerrado manualmente. A sintaxe para isso é a palavra-chave while, seguida por uma expressão que resulta em Verdadeiro ou Falso.

countdown = 3

while countdown > 0:

print(countdown, "...", sep="")

countdown -= 1

print("Blast off!")

Podemos usar um loop *while* para levar o nosso código Yoda um passo adiante e continuar pedindo para o usuário escolher até conseguirmos a escolha que queremos.

choice = None

while choice != 1:

print("Options:\n1 - Do\n2 - Do Not\n3 - Try")

choice = int(input("Choose one: "))

if choice == 1:

print("Yes. Good.\n")

elif choice == 2:

print("That is why you fail.\n")

elif choice == 3:

print("Do. Or do not. There is no Try.\n")

else:

print("You seek Yoda.\n")

# 1. Enumerable Magic #3 - Does My List Include This?

Crie uma função que aceite uma lista e um item, e retorne **True** caso o item pertença à lista, e **False** caso não pertença.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/545991b4cbae2a5fda000158/train/python)

**Exemplo**

include([1,2,3,4,5], 5) # Retorno -> True

include([2,4,6,8,10], 12) # Retorno -> False

# 2. Count the Monkeys!

Você levou seu filho para a floresta para ver os macacos. Você sabe que existe um certo número deles (n), mas seu filho é muito jovem para apreciar o número total, ele precisa começar a contar a partir do 1.

Como um bom pai, você senta e começa a contar com ele. Dado um número (n), popule uma lista com todos os números até n incluindo o próprio n, mas excluindo o zero.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/56f69d9f9400f508fb000ba7)

**Exemplo**

monkeyCount(10) # Retorno -> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

monkeyCount(1) # Retorno -> [1]

Funções

* Função é um bloco de código que só vai ser executado quando chamado.
* Uma função também pode retornar dados.
* Para executar uma função, coloque o nome (da função) seguido de parênteses.

Exemplo

def funcao():

print('dentro da função')

funcao() # <- Executando a função

# Retorno

> dentro da função

Parâmetros

* Uma função pode receber múltiplos argumentos nos parâmetros dela ou não receber nenhum.
* Os argumentos são passados nos parênteses, que estão logo após o nome da função.
* Você também pode passar alguns parâmetros padrão para sua função.

Exemplo

Utilizando apenas um argumento.

def funcao(nome):

print('Bom dia, ' + nome)

funcao('Kenzie!')

funcao('Luan')

funcao('Guilherme')

# Retornos

> Bom dia, Kenzie!

> Bom dia, Luan

> Bom dia, Guilherme

Exemplo 2

Utilizando múltiplos argumentos.

def funcao(nome1, nome2, nome3):

print(nome1 + ', ' + nome2 + ', ' + nome3)

funcao('Seiya', 'Hyoga', 'Ikki')

funcao('Naruto', 'Sasuke', 'Sakura')

funcao('Inuyasha', 'Kagome', 'Miroku')

# Retornos

> Seiya, Hyoga, Ikki

> Naruto, Sasuke, Sakura

> Inuyasha, Kagome, Miroku

Parâmetros Padrão e Retorno

* Quando você chama uma função sem nenhum argumento, esse parâmetro padrão sera rodado no lugar.
* Você pode retornar o valor de uma função para armazenar em uma variável.

Exemplo

def funcao(pais = 'Brasil'):

return pais

argentina = funcao('Argentina')

brasil = funcao()

chile = funcao('Chile')

print(argentina, brasil, chile)

# Retorno

> Argentina Brasil Chile

1. Function 1 - hello world

Faça uma função simples chamada greet que retorna o mais que famoso 'hello world!'

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/523b4ff7adca849afe000035/train/python)

**Exemplo**

greet() # Retorno -> 'hello world'

2. Function 2 - squaring an argument

Agora voce tem que escrever uma função que recebe um argumento (n) e retorne o quadrado dele.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/523b623152af8a30c6000027/train/python)

**Exemplo**

square(2) # Retorno -> 4

square(5) # Retorno -> 25

square(10) # Retorno -> 100

3. Function 3 - Multiplying two numbers

Implemente uma função multiply que receba 2 números e retorne a multiplicação deles.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/523b66342d0c301ae400003b/train/python)

**Exemplo**

multiply(4, 10) # Retorno -> 40

multiply(5, 2) # Retorno -> 10

multiply(2, 100) # Retorno -> 200

4. Grasshopper - Basic Function Fixer

**Arrume a função**  
Eu criei essa função para adicionar 5 a qualquer numero que for passado para ela e retornar o novo valor. Ela não sinaliza nenhum nenhum erro, mas retorna o número.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/56200d610758762fb0000002/train/python)

**Exemplo**

add\_five(5) # Retorno -> 10

add\_five(-10) # Retorno -> -5

5. Grasshopper - Function syntax debugging

Um estudante estava trabalhando em uma função e cometeu alguns erros de sintaxe enquanto codava. Ajude-o a encontrar seus erros e arrume-os.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/56dae9dc54c0acd29d00109a/train/python)

**Exemplo**

main('pegar ', 'item') # Retorno -> 'pegar item'

main('Zero ao', ' Python') # Retorno -> 'Zero ao Python'

6. Grasshopper - Summation

* Escreva um programa que encontre a somatória de cada número entre 1 e o num.
* O número (num) sempre será um inteiro positivo maior que zero.

**Para realizar essa prática utilize este**[**Link do Codewars.**](https://www.codewars.com/kata/55d24f55d7dd296eb9000030/train/python)

**Exemplo**

summation(2) # Retorno -> 3 (1 + 2)

summation(5) # Retorno -> 15 (1 + 2 + 3 + 4 + 5)

summation(8) # Retorno -> 36 (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8)

**Boa diversão, dev!! 😄**

Dicionários

* São utilizados para guardar dados em chave:valor.
* Não aceitam duplicatas.

Exemplo

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur"

}

print(meu\_nome)

# Retorno

> {'nome': 'Rivon', 'sobrenome': "L'mour Calcur"}

Itens de Dicionário

* Itens de dicionário são os pares de chave:valor.

Exemplo

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur"

}

print(meu\_dicionario['nome'])

# Retorno

> Rivon

Duplicatas não são aceitas

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 30,

"idade": 22

}

print(meu\_dicionario)

# Retorno

> {'nome': 'Rivon', 'sobrenome': "L'mour Calcur", 'idade': 22}

Itens de Dicionários - Tipos de Dado

* Os valores de um dicionário pode receber qualquer tipo de dado.

Exemplo

meu\_dicionario = {

"marca": "Ford", # <- Tipo string

"eletrico": False, # <- Tipo booleano

"ano": 1964, # <- Tipo inteiro

"cors": ["red", "white", "blue"] # <- Tipo lista

}

print(meu\_dicionario)

# Retorno

> {'marca': 'Ford', 'eletrico': False, 'ano': 1964, 'cors': ['red', 'white', 'blue']}

Métodos de dicionário

.keys()

* Retorna uma lista com as chaves do dicionário.

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 22

}

print(meu\_dicionario.keys())

# Retorno

> dict\_keys(['nome', 'sobrenome', 'idade'])

.values()

* Retorna uma lista com as valores do dicionário.

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 22

}

print(meu\_dicionario.keys())

# Retorno

> dict\_values(['Rivon', "L'mour Calcur", 22])

.items()

* Retorna uma lista de tuplas contendo os pares de chave:valor do dicionário.

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 22

}

print(meu\_dicionario.items())

# Retorno

> dict\_items([('nome', 'Rivon'), ('sobrenome', "L'mour Calcur"), ('idade', 22)])

Iteração em Dicionário

* Podemos fazer uma iteração em um dicionário utilizando um for, em outras palavras, vamos percorrer esse dicionário.

Percorrendo um Dicionário

* Por padrão ele percorre pelas chaves do dicionário.

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 22

}

for chave in meu\_dicionario:

print(chave)

# Return

> nome

> sobrenome

> idade

Percorrendo utilizando .keys()

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 22

}

for chave in meu\_dicionario.keys():

print(chave)

# Retorno

> nome

> sobrenome

> idade

Percorrendo utilizando .values()

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 22

}

for valor in meu\_dicionario.values():

print(valor)

# Retorno

> Rivon

> L'mour Calcur

> 22

Percorrendo utilizando .items()

meu\_dicionario = {

"nome": "Rivon",

"sobrenome": "L'mour Calcur",

"idade": 22

}

for chave, valor in meu\_dicionario.items():

print(chave, valor)

> nome Rivon

> sobrenome L'mour Calcur

> idade 22

## 1. L1: Bartender, drinks!

**Para realizar essa prática utilize este**[**Link do Codewars**](https://www.codewars.com/kata/568dc014440f03b13900001d/train/python)**.**

* Sua função vai receber uma string nos parâmetros e retornar de acordo com a tabela abaixo:

|  |  |
| --- | --- |
| **Chave** | **Valor** |
| "Jabroni" | "Patron Tequila" |
| "School Counselor" | "Anything with Alcohol" |
| "Programmer" | "Hipster Craft Beer" |
| "Bike Gang Member" | "Moonshine" |
| "Politican" | "Your tax dollars" |
| "Rapper" | "Cristal" |
| *qualquer outra coisa* | "Beer" |

***Nota:****qualquer outra coisa*é um caso base: se a string passada por parâmetro não existir na nossa tabela, retornar 'Beer'.

Atente-se em cobrir os casos em que certas palavras não vêm com a formatação correta, exemplo: o parâmetro "pOLitiCIaN" ainda deve retornar "Your tax dollars".

**Exemplo**

get\_drink\_by\_profession("perSON") # Retorno -> "Beer"

get\_drink\_by\_profession("jabrOni") # Retorno -> "Patron Tequila"

get\_drink\_by\_profession("bike ganG member") # Retorno -> "Moonshine"

get\_drink\_by\_profession("Programmer") # Retorno -> "Hipster Craft Beer"

Hora da Prática!

Para praticar os conceitos aprendidos sobre Iteração de Dicionário vamos fazer um exercício. 😀

**OBS:**

* Crie um arquivo chamado main.py

Exercício

* **Parâmetros**
  + Sua função receberá um dicionário no parâmetro.
* **Procedimento**
  + Criar uma lista;
  + Se a quantidade e o valor for **menor que 7**, deve criar um dicionário com a **chave sendo o nome da fruta** e o **valor sendo a quantidade** dela e adicionar esse dicionário a lista criada;
* **Retorno**
  + Deve retornar uma lista de dicionários contendo o **nome da fruta**como chave e a **quantidade comprada** como valor.

Exemplo

frutas = {

"melancia": {"quantidade": 4, "preco": 10},

"pera": {"quantidade": 2, "preco": 3},

"uva": {"quantidade": 8, "preco": 8},

"ameixa": {"quantidade": 5, "preco": 2},

"abacaxi": {"quantidade": 15, "preco": 4},

"banana": {"quantidade": 6, "preco": 4}

}

compras = comprar\_frutas(frutas)

print(compras)

# Retorno

> [{'pera': 2}, {'ameixa': 5}, {'banana': 6}]

|  |  |  |
| --- | --- | --- |
| **Assinatura da Função** | **Procedimento** | **Esperado que** |
| comprar\_frutas(frutas) | Criar uma lista;  Armazenar em uma lista de dicionários com o **nome** e a **quantidade** das frutas que tenham **valor e quantidade menor que 7** | Retorne uma lista de dicionários |

# Utilizando dados de um arquivo CSV

## Persistência de dados

Persistência, em computação, refere à característica de um dado sobreviver ao processo que o criou. Como exemplo, podemos utilizar várias formas de termos persistência em uma aplicação, como em arquivos, bancos de dados, bancos em memória RAM, entre outros.

## O que é um arquivo CSV

Um arquivo CSV organiza dados de maneira tabular, sendo possível distinguirmos linhas e colunas, onde as colunas são divididas por uma vírgula simples.

Esta organização de dados é semelhante ao que vemos em um arquivo do famoso Excel ou do Google Spreadsheet.

Vejamos um exemplo abaixo, que armezana informações de pães de um estabelecimento:

paes.csv

id\_produto,descricao,valor,desconto

1,Pão de Queijo,1.00,0.0

2,Pão de Forma,6.00,0.4

3,Pão de Milho,0.70,0.0

4,Pão de Mel,2.50,0.3

Processando um arquivo CSV

E aqui, mais uma vez o Python nos trás uma funcionalidade, o módulo csv que é um *built-in*, que nos possibilita realizar operações com arquivos CSV

Lendo um arquivo CSV

Para fazer esta leitura vamos instruir a máquina a fazer o seguinte

* importar o módulo csv
* abrir o arquivo csv em modo de leitura
* criar um leitor de csv, que é um iterador sobre o arquivo
* iterar sobre as linhas para obtermos os dados
* fechar o arquivo csv

Nosso código equivalente é o seguinte

import csv

with open('paes.csv', 'r') as paes\_csv:

reader = csv.DictReader(paes\_csv)

for dic in reader:

print(dic)

A saída (output) será a seguinte

{'id\_produto': '1', 'descricao': 'Pão de Queijo', 'valor': '1.00', 'desconto': '0.0'}

{'id\_produto': '2', 'descricao': 'Pão de Forma', 'valor': '6.00', 'desconto': '0.4'}

{'id\_produto': '3', 'descricao': 'Pão de Milho', 'valor': '0.70', 'desconto': '0.0'}

{'id\_produto': '4', 'descricao': 'Pão de Mel', 'valor': '2.50', 'desconto': '0.3'}

Escrevendo em arquivo CSV

Para incluir um novo registro no nosso CSV de pães vamos ler o nosso arquivo csv utilizando o DictReader e então adicionar um novo elemento a lista e então adicioná-la no fim do arquivo, instruindo a máquina a fazer o seguinte:

* importar o módulo csv
* abrir o arquivo em modo leitura
* criar um escritor de csv
  + definir o caractere de delimitador
  + definir o caractere fieldnames
* criar uma lista com os dados para serem colocados na linha
* passar a lista para o escritor adicioná-la no fim do arquivo csv
* fechar o arquivo csv

Tranformando isso em código Python, temos o seguinte:

import csv

# Essa função faz a leitura de todos os dicionários dentro do arquivo 'paes.csv', não precisa se preocupar, você já fez anterioremente.

def all\_products(FILENAME):

with open('paes.csv', 'r') as paes\_csv:

return [dic for dic in csv.DictReader(paes\_csv)]

# Aqui estamos adicionando o nosso novo dicionário na nossa lista de produtos.

pao = {'id\_produto': 5, 'descricao': 'Pão cevado', 'valor': 4.00, 'desconto': 0.0}

all\_products\_list = all\_products('paes.csv')

all\_products\_list.append(pao)

# E aqui fazemos a escrita do nosso arquivo

with open('paes.csv', 'w') as paes\_csv:

writer = csv.DictWriter(paes\_csv, delimiter=',', fieldnames=['id\_produto', 'descricao', 'valor', 'desconto'])

writer.writeheader()

for product in all\_products\_list:

writer.writerow(product)

Dessa forma teremos uma nova linha no nosso arquivo paes.csv, como a seguir.

id\_produto,descricao,valor,desconto

1,Pão de Queijo,1.00,0.0

2,Pão de Forma,6.00,0.4

3,Pão de Milho,0.70,0.0

4,Pão de Mel,2.50,0.3

5,Pão cevado,4.0,0.0

Resources:

* [Documentação do Python](https://docs.python.org/pt-br/3/library/csv.html)
  + [DictReader](https://docs.python.org/pt-br/3/library/csv.html#csv.DictReader)
  + [DictWriter](https://docs.python.org/pt-br/3/library/csv.html#csv.DictWriter)

**Mathplotlib**

Vamos dar uma olhada na biblioteca de geração de gráficos matplotlib.

Instalação

Utilize o seguinte comando no seu terminal para instalar a biblioteca.

pip install matplotlib

**Caso esteja utilizando o google colab não precisa fazer essa instalação.**

**Utilizando**

Se você estiver utilizando o google colab basta incluir em sua célula.

Caso você esteja fazendo localmente basta importar a biblioteca em seu arquivo main.py.

import matplotlib.pyplot as plt

***NOTA*:**você está importando sua biblioteca e apelidando ela de plt para poder utilizar em seu código.

Agora que importamos a biblioteca vamos fazer uma função de criação de gráfico:

def criar\_grafico(dado):

plt.bar(list(data.keys()), list(data.values()))

plt.xlabel('chave')

plt.ylabel('valor')

plt.savefig('grafico\_de\_barra.png')

**Nota:**Utilizamos list(data.keys()) e list(data.values()), para transformarmos o retorno do .keys()/.values() no tipo lista, por padrão ele ira retornar como tipo dict\_keys, para o .keys, e dict\_values, para o .values, então para evitar alguns possíveis erros, vamos fazer essa conversão.

**Destrinchando a função**

1. **plt.bar:**está recebendo duas listas por parâmetro, sendo a primeira as chaves, que ficaria no **eixo x** do nosso gráfico, e a segunda sendo os valores, que ficaria no **eixo y** do gráfico.
2. **plt.xlabel:**está recebendo a string '**chave**', que será o nome **do eixo x**.
3. **plt.ylabel:**está recebendo a string '**valor**', que será o nome **do eixo y**.
4. **plt.savefig:**está recebendo uma string '**grafico\_de\_barra.png**', que será o nome da imagem do nosso gráfico.

**Executando a função**

meu\_dicionario = {

'chave1': 1,

'chave2': 2,

'chave3': 3,

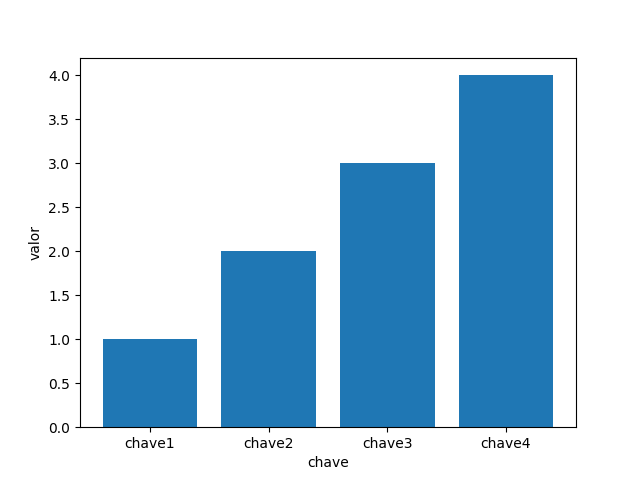
'chave4': 4

}

criar\_grafico(meu\_dicionario)

Gráfico de Barra

**Nota:**a imagem será criada no mesmo diretório em que sua função foi executada.



Fazendo um Gráfico de Linha

Vamos fazer apenas duas alterações no nosso código!

def criar\_grafico(dado):

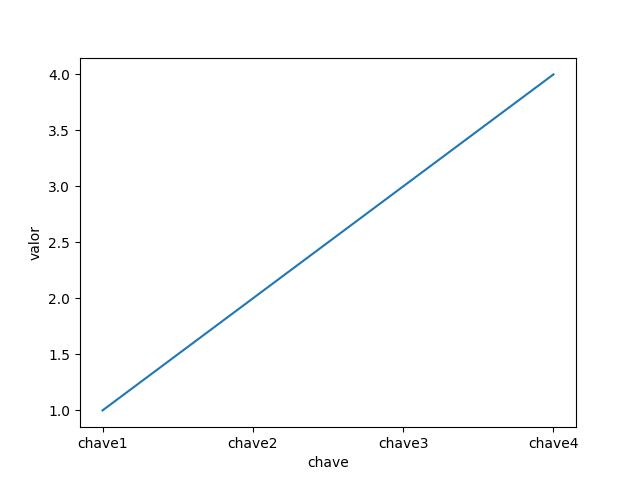
plt.plot(list(data.keys()), list(data.values())) # mudamos aqui de 'bar' para 'plot'

plt.xlabel('chave')

plt.ylabel('valor')

plt.savefig('grafico\_de\_linha.png') # e aqui de 'barra' para 'linha'

**Gráfico de Linha**



Hora da Prática!

Para praticar os conceitos aprendidos sobre CSV vamos fazer alguns exercícios de processamento de CSV 😀

**OBS:**

* Crie um arquivo chamado main.py
* Crie um arquivo CSV chamado characters.csv
* Os fieldnames para o arquivo CSV que será utilizado deve ser:

id,name,intelligence,power,strength,agility

Exercício 1

|  |  |
| --- | --- |
| Assinatura da Função | read\_character\_list(filename) |
| Procedimento | Abrir o arquivo em modo de leitura;  Criar um objeto leitor de CSV a partir do arquivo aberto;  Retornar uma lista de tuplas, em que cada tupla é um registro do CSV. Não incluir o cabeçalho do CSV. Cada um dos valores numéricos devem ser retornados no formato int |

Exercício 2

|  |  |
| --- | --- |
| Assinatura da Função | read\_specific\_character(filename, character\_id) |
| Procedimento | Abrir o arquivo em modo de leitura;  Criar um objeto leitor de CSV a paritr do arquivo aberto;  Procurar pelo registro com id idêntico ao parâmetro character\_id recebido;  Retornar um dicionário em que as chaves são iguais ao cabeçalho do CSV e os valores iguais ao registro do character encontrado |

Exercício 3

|  |  |
| --- | --- |
| Assinatura da Função | create\_character(filename, name, intelligence, power, strength, agility) |
| Procedimento | Abrir o arquivo em **modo de escrita com a flag a;**  Criar um objeto leitor de CSV a paritr do arquivo aberto;  Criar um id para o novo registro;  Adicionar uma nova linha no CSV com o novo registro; |