**Apostila de Estrutura de Dados**

Sumário

[Introdução 3](#_Toc73453897)

[Python 4](#_Toc73453898)

[Ferramentas de Código Python 4](#_Toc73453899)

[Módulo de Tratamento de Exceção 4](#_Toc73453900)

[Tipos de Erros no Python 4](#_Toc73453901)

[Tratamento de Erros 5](#_Toc73453902)

[Tratamento de Diversos Erros 6](#_Toc73453903)

[Tratamento Infinito dos Erros até a Saída 6](#_Toc73453904)

[Leitura e Gravação em Arquivo de Texto 7](#_Toc73453905)

[Leitura 7](#_Toc73453906)

[Alocar o Texto em uma Lista 7](#_Toc73453907)

[Gravação em um Novo Arquivo de Texto 7](#_Toc73453908)

[Exercício da Leitura e Gravação 9](#_Toc73453909)

[Expressões Regulares 10](#_Toc73453910)

[Metacaracteres 10](#_Toc73453911)

[Quantificadores 11](#_Toc73453912)

[Exemplos 11](#_Toc73453913)

[Expressões regulares no Python 12](#_Toc73453914)

[Método Search 12](#_Toc73453915)

# Introdução

Quando possuímos um problema e precisamos criar uma estrutura de dados e armazenar as informações para resolver.

Quando se tem um problema para resolver, as perguntas devem ser feitas:

* **Como representar** esses problemas em computadores?
* **Como construir** os algoritmos necessários?
* Que **estrutura** de **dados** utilizar?
* De que **forma** os dados estarão organizados?
* Quais **operações** podem ser realizadas nestes dados?

# Python

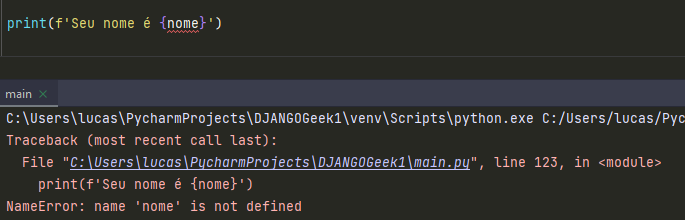
## Ferramentas de Código Python

Spider (Ciência de Dados e IA), Jupyter Notebook(Visualizações de Dados, Gráficos e Relatórios), PyCharm e o Google Colabs (programação online).

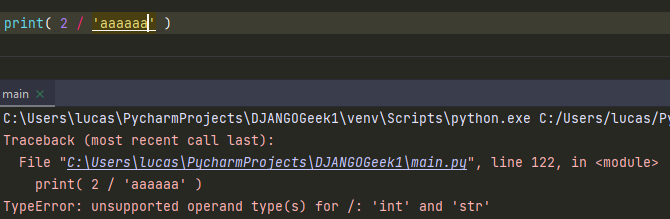
## Módulo de Tratamento de Exceção

### Tipos de Erros no Python

**NameError**: Variável não definida

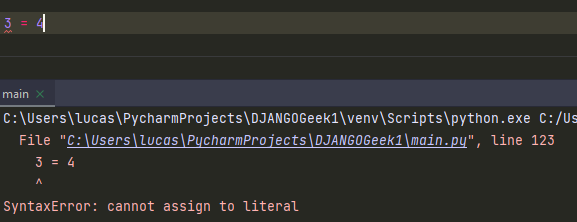


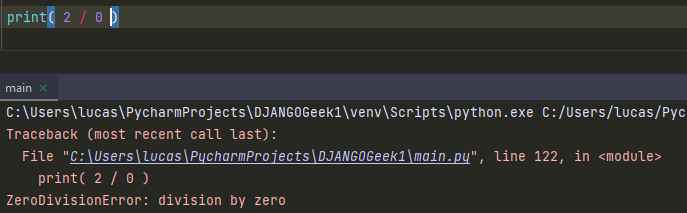
**TypeError**: Tipo de dados Incompatíveis (Dividir número por string)

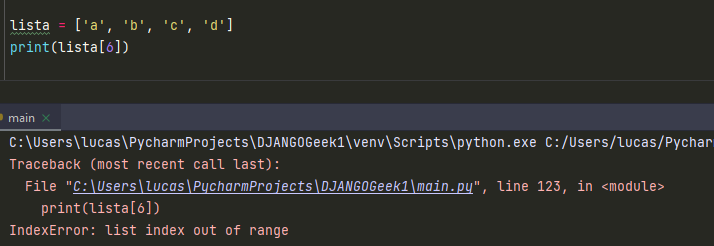


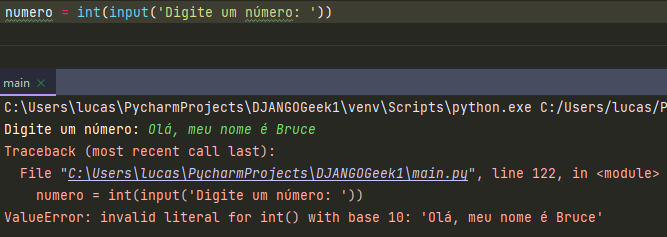
**RunTimeError**: Erro de Execução

**SyntaxError**: Erro de sintaxe, sintaxe inválida e não reconhecida pelo interpretador



**ZeroDivisionError**: Erro de divisão por zero

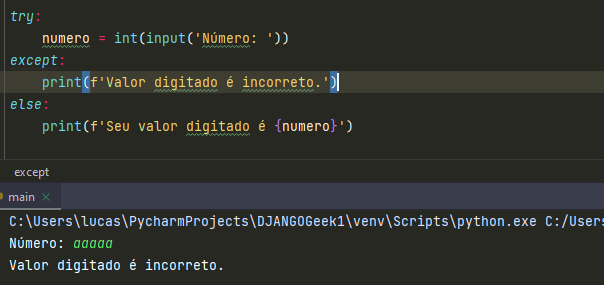
**IndexError**: Índice está fora da coleção (listas, vetores, matrizes)

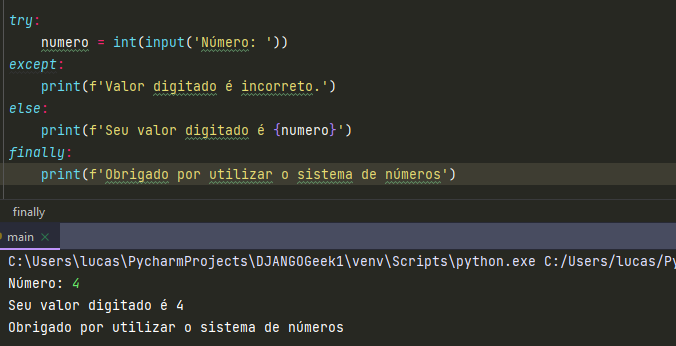
**ValueError**: Erro de invalidez do tipo de dado (Solicitado int e recebido string)

**EOFError**:

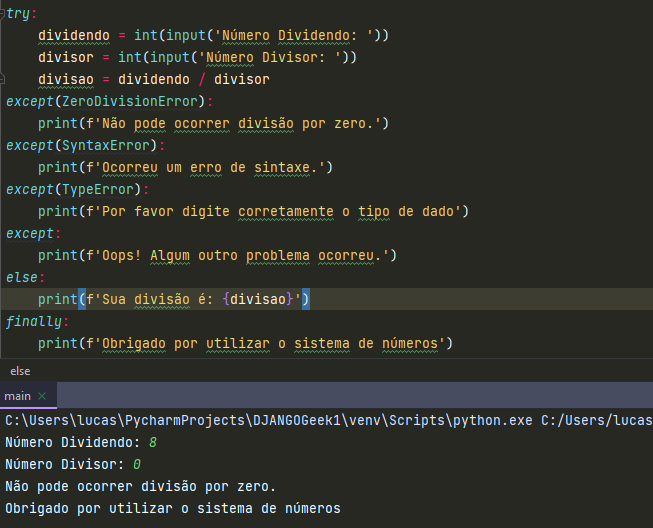
### Tratamento de Erros

Através de um bloco: Try, Except

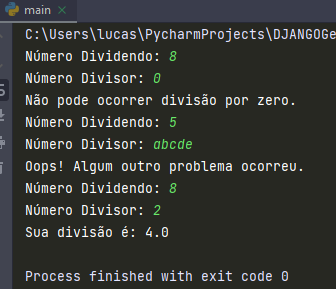
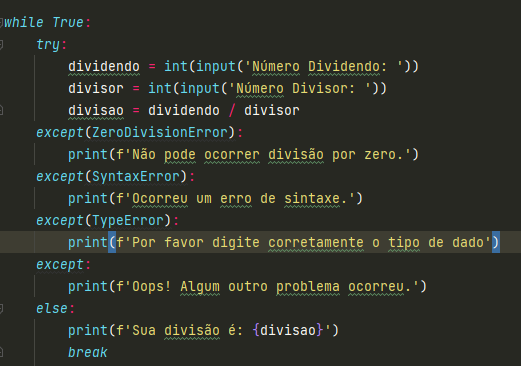


Bloco Try, Except, Else e Finally.

### Tratamento de Diversos Erros



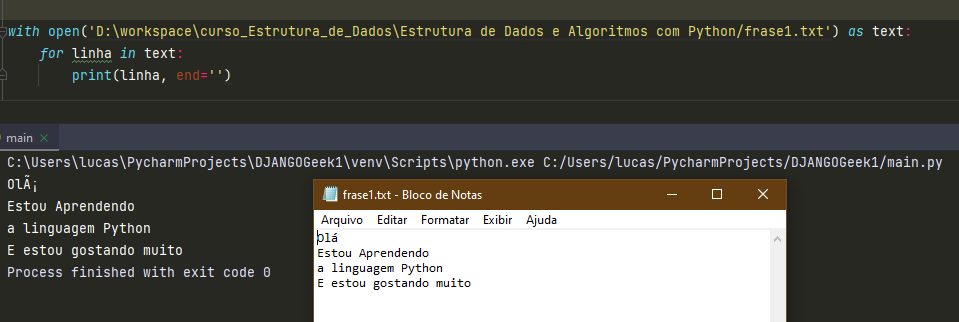
### Tratamento Infinito dos Erros até a Saída



## Leitura e Gravação em Arquivo de Texto

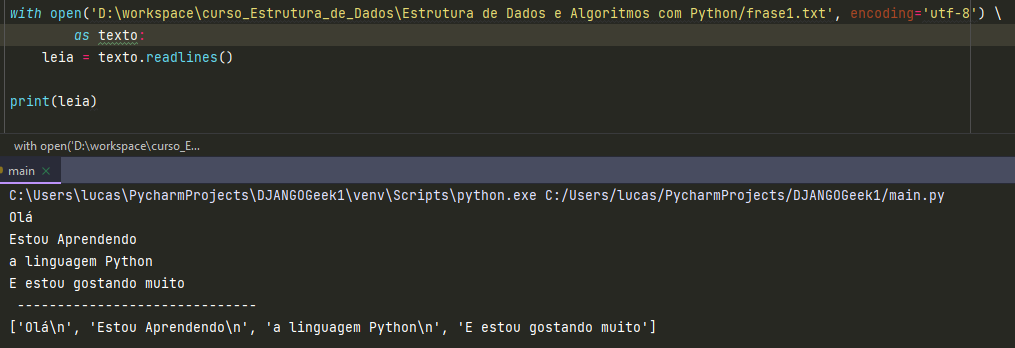
### Leitura

Com o comando ‘’with open(‘caminho\_do\_arquivo/nome\_do\_arquivo’) as nome\_variável”

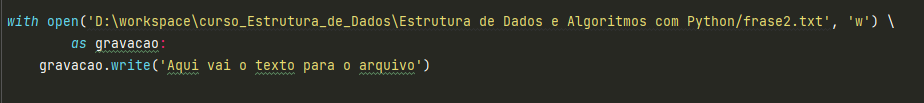
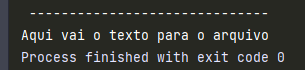
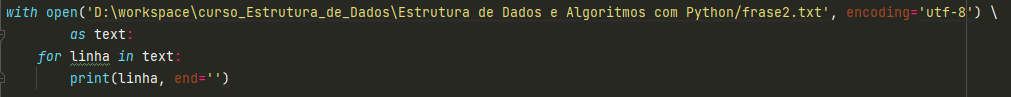
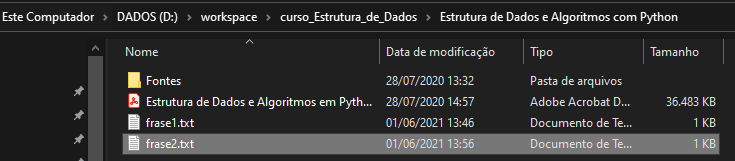


### Alocar o Texto em uma Lista

Através de uma passagem para uma variável que vai receber a variável do arquivo e a chamada da função “readlines()”.



### Gravação em um Novo Arquivo de Texto

Agora o arquivo foi gravado e criado no local direcionado. 

**Código do arquivo que fez as leituras e gravações:**

with open('D:\workspace\curso\_Estrutura\_de\_Dados\Estrutura de Dados e Algoritmos com Python/frase1.txt', encoding='utf-8') \

as text:

for linha in text:

print(linha, end='')

print('\n', '-' \* 30)

with open('D:\workspace\curso\_Estrutura\_de\_Dados\Estrutura de Dados e Algoritmos com Python/frase1.txt', encoding='utf-8') \

as texto:

leia = texto.readlines()

print(leia)

print('\n', '-' \* 30)

with open('D:\workspace\curso\_Estrutura\_de\_Dados\Estrutura de Dados e Algoritmos com Python/frase2.txt', 'w') \

as gravacao:

gravacao.write('Aqui vai o texto para o arquivo')

print('\n', '-' \* 30)

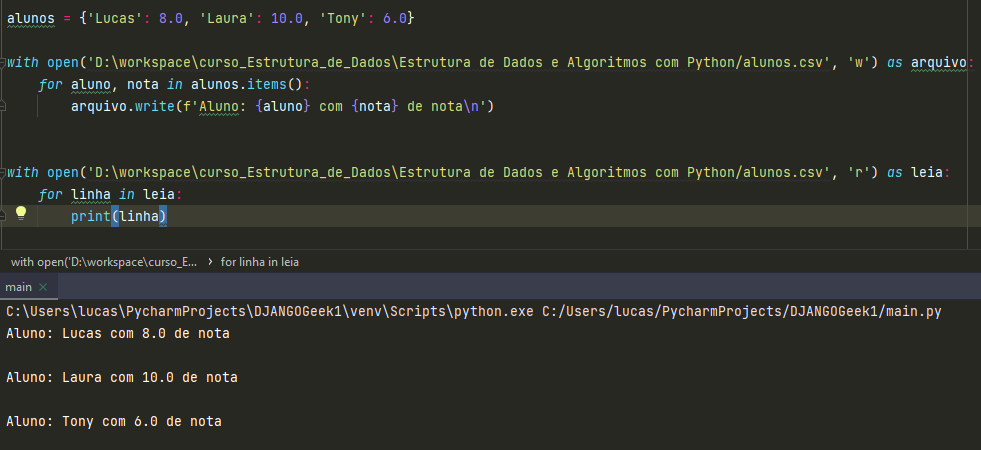
with open('D:\workspace\curso\_Estrutura\_de\_Dados\Estrutura de Dados e Algoritmos com Python/frase2.txt', encoding='utf-8') \

as text:

for linha in text:

print(linha, end='')

### Exercício da Leitura e Gravação



## Expressões Regulares

Busca “sofisticada” por padrões em variáveis do tipo string. Por exemplo, variáveis que comecem ou terminem com uma determinada letra (‘a’ ou ‘b’).

Procurar em e-mails, números de telefones, CEPs.

Por exemplo a frase: “*O mundo está mudando. Tecnologias utilizadas na atualidade estão tornando as tarefas cada vez mais mecanizadas e práticas, onde a mente humana é deixada livre para exercer o papel criativo*”.

1. Extrair todas as palavras que começam com vogais e terminam com consoante.
2. Extrair todas as palavras que possuem de 5 a 9 letras.
3. Extrair todos os trechos separados por ponto(“.”).
4. Extrair todas as palavras que começam com “c”, possuem “r”, “e” ou “u” no meio e terminam com “a”, “i” ou “x”.

Método **Search**: Encontra posições de padrões **dentro** de uma string se estiverem presentes

Método **Match:** Encontra se o **começo** de uma string é igual a um determinado padrão

Método **Findall:** Encontra todas as *substrings*(pedaço de uma string) em uma string que correspondem a um padrão. \***Substring**: “corresp” é uma substrings da string “correspondem”.

### Metacaracteres

Qualquer caractere: (**.**) (exceto linha nova)

Qualquer caractere **alfanumérico**: **\w** (somente letras)

Qualquer caractere **não**-alfanumérico: **\W** (caracteres especiais)

Qualquer caractere que seja um dígito: **\d**

Qualquer caractere que não seja um dígito: **\D**

Espaços em Branco: **\s**

Começa com determinado caractere: **^** - ex: ^a – começa com ‘a’.

Terminar com determinado caractere: **$** - ex: $a – termina com ‘a’.

Antes de metacaracteres: **\** - Usado para significado literal “\.”

### Quantificadores

Determinam quantas vezes os metacaracteres aparecem

**[ ]** – Quantas vezes os que estão dentro dos colchetes, ex: [a,b]

**( )** – Captura grupos de caracteres, ex: (0, 9) ou (a,z)

**\*** - De zero a mais vezes(sempre ao caractere anterior)

**?** – Zero ou uma vez (sempre ao caractere anterior)

**+** - Uma ou mais vezes (sempre ao caractere anterior)

**{m, n}** – De m a n vezes, ex: {2,2}

**|** - Ou

### Exemplos

**lucas@gmail**.**com**  
**laura@gmail**.**com**  
**tony@yahoo**.**com.br**

Expressão regular que detecta e-mail: **\w**+**@\w**+\.**w+**

(Lendo: Precisa iniciar com caracteres alfanuméricos (vermelho) ai vem o “+” que precisa ser um ou mais caracteres especiais que seja o “@”, seguindo por novamente caracteres alfanuméricos(roxo), e que seja seguido com o “+” que precisa ser um ou mais caractere especial “.”, sendo necessário a barra “\.” para indicar que trata-se de um metacaractere ponto e não todo, e a seguido novamente por mais caracteres alfanuméricos).

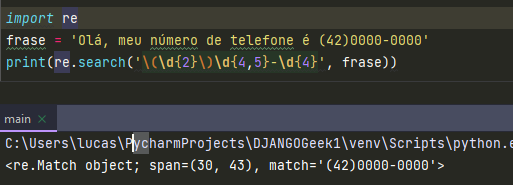
Atenção: o Quantificador (+) após o “\w” significa que será um conjunto de caracteres alfanuméricos que seguirão.

### Expressões regulares no Python

Necessário importar o pacote “re” – Regular Expression no Python.

### Método Search

1º Exemplo: **frase** = 'Olá, meu número de telefone é (**42**)**0000**-**0000**'  
Código Python: re.search('\(**\d{2}**\)**\d{4,5}**-**\d{4}**', frase).  
Lendo: Primeiro passamos o “\(“ que significa a expressão literal para buscar um “(“ , e depois procure qualquer caractere que seja um dígito “\d” e que sejam dois números {2} e depois novamente a expressão literal de “\)” para buscar um “)”. E depois mais “\d” para buscar dígitos e que seja de 4 a 5 dígitos “{4,5}”, e o hífen, não é necessário metacaractere, e depois novamente buscar mais 4 caracteres numéricos. “\d{4}”

O retorno no console:   
  
O “**span**” significa que o retorno foi a partir do caractere na posição “30” e o final foi na posição “43”. Onde o **Match** para esse tipo de Padrão de uma Expressão regular, foi o (42)0000-0000.

2º Exemplo: frase = A placa do carro que eu anotei durante o acidente foi **FRT**-**1998**  
Código Python: re.search(‘**[A-Z]{3}**-**\d{4}**', frase)

Cuidado porque o conjunto de caracteres está buscando por A-Z maiúsculos, caso seja digitado qualquer caractere minúsculo, ocorrerá um erro. Por isso, pode usar:   
**[A-Za-z]{3}**-**\d{4}**'

**Código** da aula: **import re**

email = 'Entre em contato, meu email é teste@teste.com'

print(re.search('\w+@\w+\.\w+', email))

frase = 'Olá, meu número de telefone é (42)90000-0000'

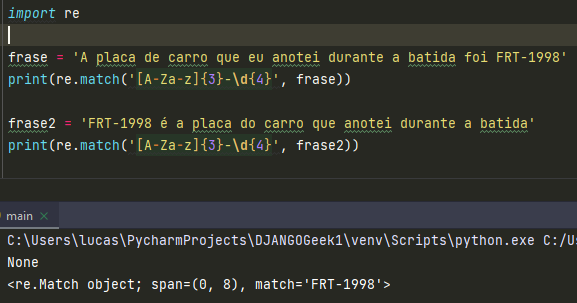
print(re.search('\(\d{2}\)\d{4,5}-\d{4}', frase))

frase = 'A placa do carro que eu anotei durante o acidente foi FRT-1998'

print(re.search('[A-Za-z]{3}-\d{4}', frase))

### Método Match

Essa função verifica se está no início, se a minha expressão regular está exatamente no início da string.



Veja que a frase1 não está no início, por isso retorna o “None”, significando que não está. Enquanto na frase2, a expressão regular está logo no início.

**Código**:  
frase = 'A placa de carro que eu anotei durante a batida foi FRT-1998'

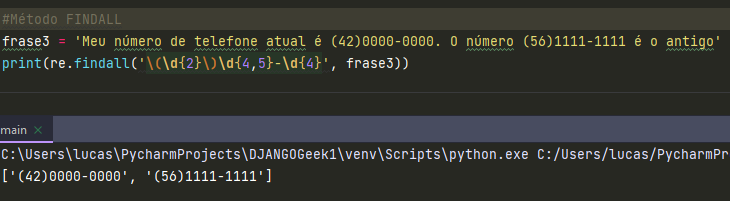
print(re.match('[A-Za-z]{3}-\d{4}', frase))

frase2 = 'FRT-1998 é a placa do carro que anotei durante a batida'

print(re.match('[A-Za-z]{3}-\d{4}', frase2))

### Método Findall

Esse método procura na string e suas substrings.



Repare que o retorno será diferente das funções anteriores, agora o retorno será em “[]”, isso significa que está em um formato de LISTA.



Também podemos utilizar para retornamos TODOS os e-mails, por exemplo. E assim percorrer uma lista.

Veja que após desenvolver a expressão regular, ela pode ser reutilizada sempre, alterando apenas o método.

# Classes

Código utilizado.

*class* Aluno:  
 *def \_\_init\_\_*(self, *nome*, *nota1*, *nota2*):  
 self.nome = *nome* self.nota1 = *nota1* self.nota2 = *nota2* self.media = 0.0  
  
 *def* calculo\_media(self):  
 self.media = (self.nota1 + self.nota2) / 2  
 *return* self.media  
  
 *def* mostra\_dados(self):  
 print(f'A nota 1: {self.nota1} \nA nota 2: {self.nota2}'  
 f'\nA média: {self.media}')  
  
  
 *def* resultado(self):  
 *if* self.media >= 6:  
 print(f'Aluno aprovado.')  
 *else*:  
 print(f'Aluno reprovado.')  
  
  
aluno1 = Aluno('Lucas', 8, 9)  
aluno2 = Aluno('Laura', 1, 3)  
  
aluno1.calculo\_media()  
aluno1.mostra\_dados()  
aluno1.resultado()  
  
aluno2.calculo\_media()  
aluno2.mostra\_dados()  
aluno2.resultado()

# Notações Big-O

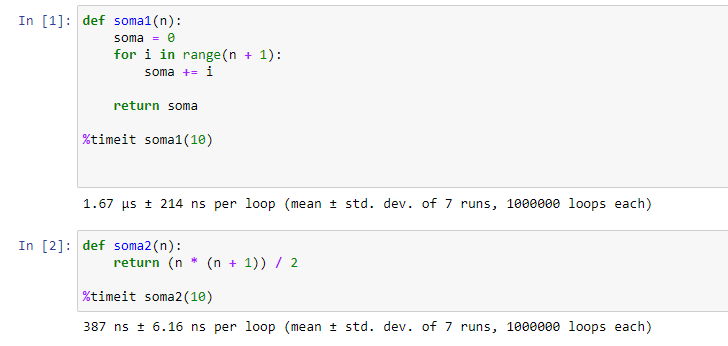
É uma notação que faz um comparativo entre algoritmos. Exemplo: Pedir para um dev desenvolver uma solução para achar a somatória de um número. E para um segundo dev achar a mesma solução. Os dois programadores realizarão funções e com base na notação Big-O, serve para comparar qual código é mais curto.

## Comparação Objetiva entre algoritmos

Calcular diferenças de processamentos.

Um dev desenvolveu pelo Google Colabs com aumento de processamento e outro em um notebook mais simples. Mesmo o segundo desenvolvedor aplicando uma função melhor, essa questão de processamento ou linguagem de programação, levando pela questão objetiva, esses pontos são desconsiderados, levando apenas a função.

**Acesse o site:** [**https://en.wikipedia.org/wiki/Orders\_of\_magnitude\_(time)**](https://en.wikipedia.org/wiki/Orders_of_magnitude_(time))



A função soma1, é considerada com “O(n)”, isso significa que ela vai levar “n” passos dependendo da quantidade de vezes que a função precisa ser chamada para executar os passos e chegar no resultado. Nessa função, ela deve percorrer os 11 passos para chegar em 55. Se passássemos 1.000.000 ao “n”, a função não seria mais escalável, o tempo de execução seria muito grande.

A função soma2, é considerada com “O(3)”, isso significa que ele passará por apenas 3 passos para chegar no mesmo número (Por exemplo, passando o n = 10, em 55). A função soma2. Se passássemos 1.000.0000 para “n”, o tempo de execução é o mesmo que se passássemos 10 para “n”. Pela construção da função.

## Observação sobre %timeit

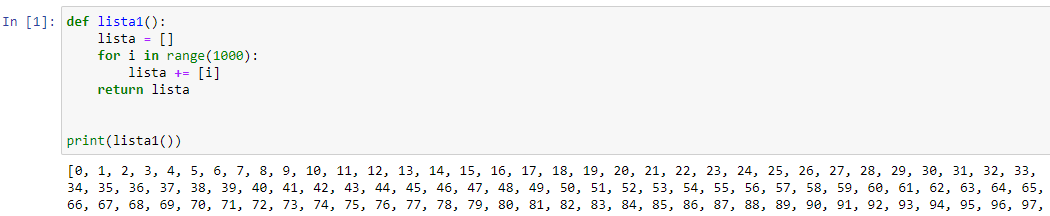
Aguardando retorno sobre questionamento da utilização de uma função no Pycharm.

Enquanto isso, faça uso da função mágica pelo Jupyter Notebook para python.  
“%timeit nome\_da\_função”

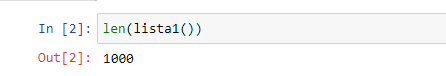
## Resumo de Big-O

Quanto mais rápido e mais eficiente for a função, menos recursos de hardware e processamento ela ocupará.

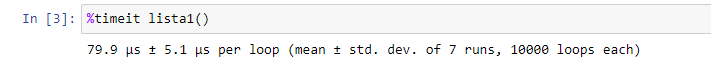
## Segundo Exemplo de Big-O



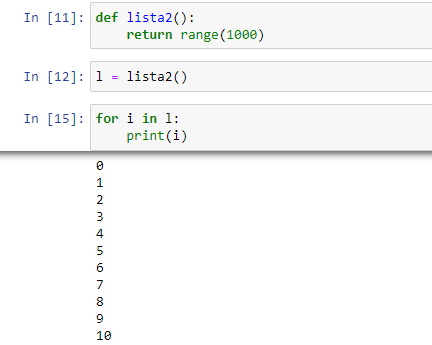
Veja que o valor está certo:



Chamando o %timeit



Criando uma segunda função lista2(), veja que ela tem o mesmo retorno que a lista1.



Esse exemplo o instrutor não foi muito feliz porque a função2 não cria uma lista e sim um range(1000)

## O que é o “n”

Na função Big-O, quando falamos que a função exerce O(n), isso significa que ela precisa que a função seja executada pelo menos n vezes para que seja executada.

## Os Tipos de Funções Big-O

**Constant**: Significa que sempre apresenta o mesmo número de passos, por exemplo a função soma2() que tem o Big-O sendo O(3)

**log(n) Logarithmic**:

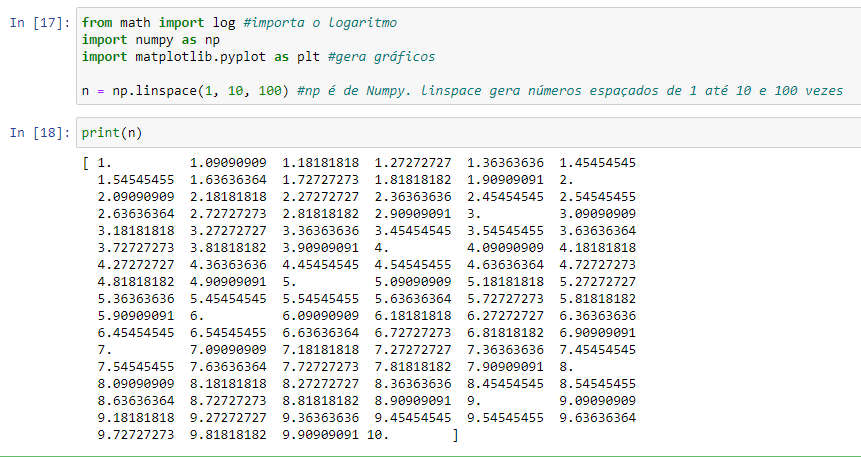
**n Linear**:

**nlog(n) Log Linear**:

**n^2 Quadratic**:

**n^3 Cubic**:

**2^n Exponential**:



Foi criado do número 1 ao número 10, criando 100 números espaçados de forma linear.

### Notações do Big-O



Criamos duas variáveis, uma que recebe a label(legenda) de cada uma das possíveis notações Big-O. E outra variável que recebe a formação da notação de cada uma das notações do Big-O.

A Constant vai criar uma quantidade de números 1 sempre igual de números conforme o tamanho (por exemplo, foram criados 100 vezes o número 1, por n é igual a 100)

A logarithmic que pelo numpy vai achar o log de cada posição de n.

A linear que é o próprio n

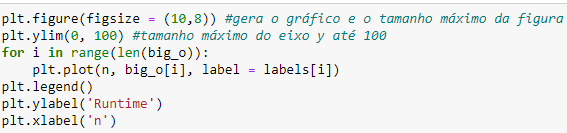
A de log linear que é o logaritmo linear de cada posição n

Quadratic e Cubic, que são cada posição de n na sua exponência

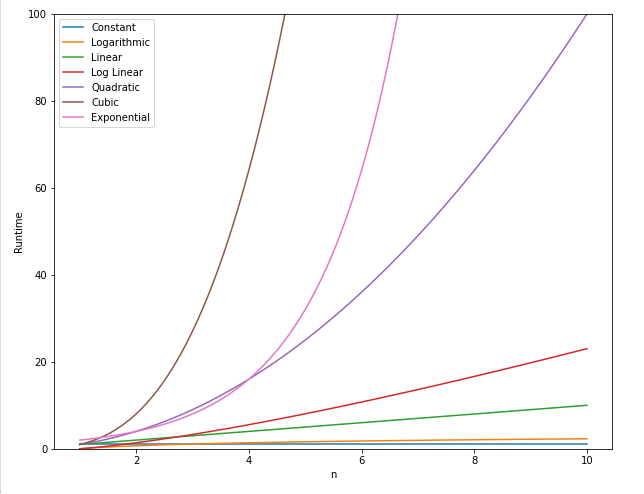
E Exponential que eleva o n a ele mesmo.

## Gerando o gráfico do Big-O

Vamos gerar um gráfico conforme a biblioteca do matplotlib



## Interpretando o gráfico de Big-O



Veja que no início, onde a quantidade de ‘n’ , onde n é a quantidade de vezes que algo precisa ser executado, ou seja, onde ‘n’ é a quantidade de passos necessários para a execução de uma função.

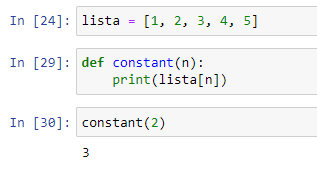
No início, onde a quantidade de vezes de uma execução é baixa, a performance é similar. Quando criamos uma função que tem passos constantes e que não é dependente da quantidade de repetições para chegar ao resultado, ela executa em um tempo constante e baixo e até mesmo a Logarithimic.

Enquanto a Linear, isso significa que ela vai aumentar o tempo de execução conforme a quantidade de passos necessários para que a função seja executada.

## Funções em Big-O

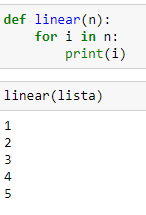
### Constante – O(1)

Vai sempre retornar o valor com apenas um único passo, executando um determinado passo.



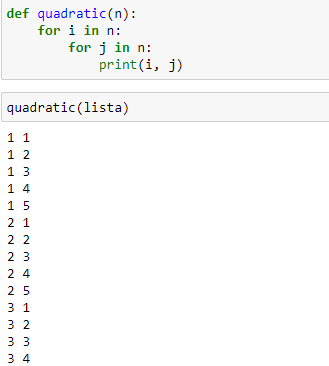
### Linear – O(n)

Como a lista tem 5 elementos, essa função vai ser executada 5 vezes. Isso gera a notação O(n) ou O(5) nesse caso. O tempo de execução aumenta proporcionalmente ao número de n.



### Quadratic – O(n^2) – Polynomial

Nesse exemplo, significa que ele vai passar pela lista até fazer todos os seus elementos, e depois vai passar novamente, até que a função seja encerrada.



# Vetores Não Ordenados

O exemplo de jogadores em um treinamento.

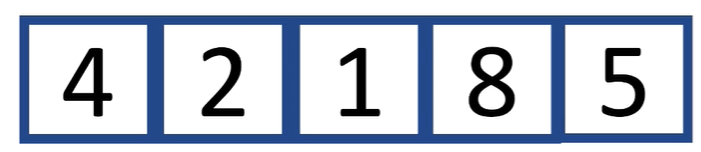
Precisamos saber quais jogadores estão treinando, quais estão descansando, quais ao chegar no gramado já são colocados para jogar, quais podem ser removidos quando vão para o descanso/casa, saber se um jogador está jogando no momento ou não.

Isso implica nas principais funções que podem ser feitas: Inserção, Pesquisa e Remoção

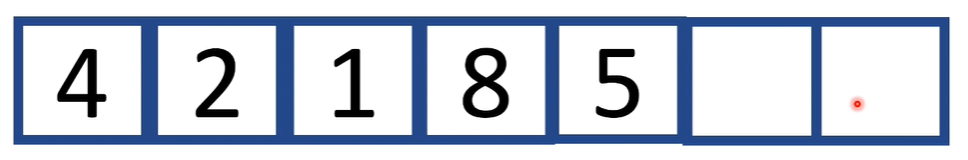


## Inserção

É inserir um dado dentro da estrutura de dados, por exemplo: O jogador 4 entrou em campo, o jogador 2 no campo, o 1 e depois o 8. Inserindo na próxima célula vaga de um vetor. Cada jogador que vai entrando vai sendo adicionado. Podemos trabalhar com um tamanho limitado ou um tamanho ilimitado (onde não se sabe a quantidade, por exemplo, uma pessoa que vai ao mercado e faz compras, não sabemos a quantidade), mas nesse caso, sabemos que só podemos ter no máximo 7 jogadores treinando por vez.



Ou



Veja que é um vetor não ordenado porque os dados são inseridos aleatoriamente no vetor, não existe um padrão.

A notação Big-O é constante, sempre O(1).

## Pesquisa Linear

Para a pesquisa, é necessário percorrer cada posição do vetor.

No melhor caso, encontraremos o número 4 já logo na posição 0.

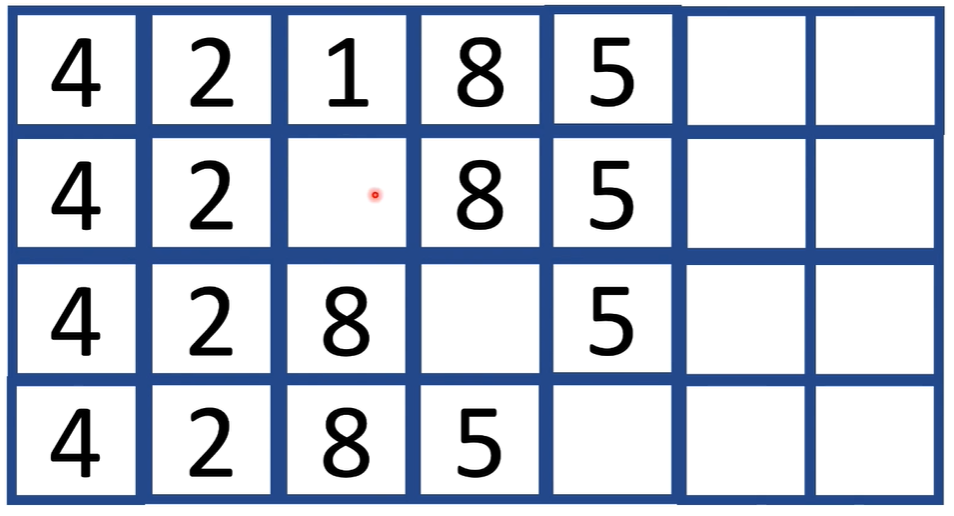
No pior caso, encontraremos o 5 que está na última posição ou que um número não existe, por exemplo o número 9 não existe.

A notação Big-O é Linear, sendo O(n), mesmo que tenha que percorrer pelo menos a metade das posições.

## Exclusão

Ao ‘’excluir’’ o número 1, não quer dizer que esse número foi excluído, ele substitui o valor 1 pelo 8, e depois no 8 pelo 5.

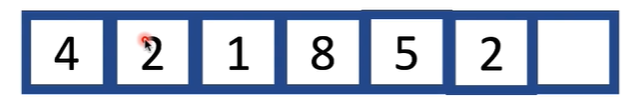
A notação Big-O será sempre O(n), pois não sabemos o número de passos da pesquisa, depois a exclusão e depois novamente o remanejamento das posições.



## Duplicatas

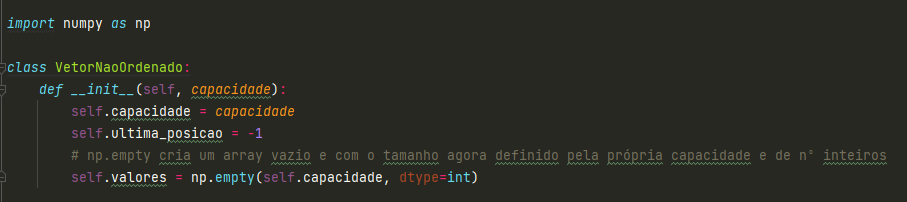
Se itens com chaves duplicados serão permitidos ou não.

A notação Big-O será O(n), para a verificação desde a primeira posição até a última, se o vetor não permitir, logo será interrompido, caso contrário, terá que percorrer todo o vetor e inserir no final. Tanto para inserir, pesquisar e excluir será O(n)



## Implementação das Funções

Crie uma classe chamada de VetorNaoOrdenado.



Ela vai receber como atributo a sua capacidade, passando o tamanho que o vetor receberá

### Inserção na classe Vetor Não Ordenado

Códígo:

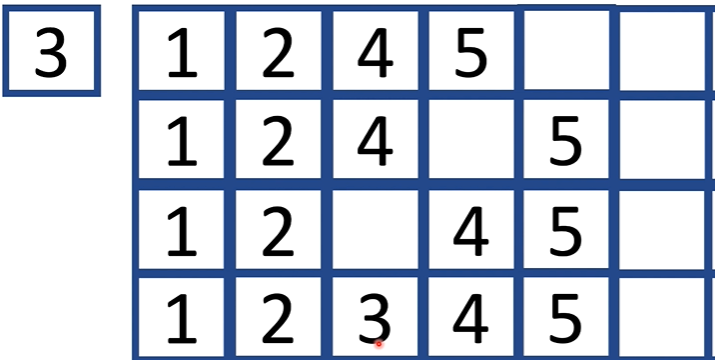
*import* numpy *as* np  
  
*class* VetorNaoOrdenado:  
 *def \_\_init\_\_*(self, *capacidade*):  
 self.capacidade = *capacidade* self.ultima\_posicao = -1  
 # np.empty cria um array vazio e com o tamanho agora definido pela própria capacidade e de nº inteiros  
 self.valores = np.empty(self.capacidade, dtype=int)  
  
 #imprime os dados do vetor  
 *def* imprime(self):  
 *if* self.ultima\_posicao == -1:  
 print(f'O vetor está vazio')  
 *else*:  
 *for* i *in* range(self.ultima\_posicao + 1):  
 print(f'Na posição [{i}]', '=', self.valores[i])  
  
  
 *def* imprimir(self):  
 print(self.valores)  
  
 #insere dados no vetor:  
 *def* inserir(self, *valor*):  
 *if* self.ultima\_posicao == self.capacidade - 1:  
 print(f'Capacidade Máxima já foi atingida.')  
 *else*:  
 self.ultima\_posicao += 1  
 self.valores[self.ultima\_posicao] = *valor  
  
  
 def* pesquisar(self, *valor*):  
 *for* i *in* range(self.ultima\_posicao + 1):  
 *if valor* == self.valores[i]:  
 *return* i #retorna qual posição o nr pesquisado está  
 *return* -1 #elemento não existe  
  
  
 *def* excluir(self, *valor*):  
 #encontra se o valor existe:  
 posicao = self.pesquisar(*valor*)  
 #apaga se encontrar:  
 *if* posicao == -1:  
 *return* -1 #não existe  
 *else*:  
 *for* i *in* range(posicao, self.ultima\_posicao):  
 self.valores[i] = self.valores[i + 1]  
  
 self.ultima\_posicao -= 1  
  
  
vetor = VetorNaoOrdenado(5)  
  
vetor.inserir(3)  
vetor.inserir(5)  
vetor.inserir(8)  
vetor.inserir(1)  
vetor.inserir(7)  
vetor.imprime()  
  
print('-' \* 30)  
  
vetor.excluir(5)  
vetor.imprime()  
#vetor.imprime()  
  
print('-' \* 30)  
  
print(f'Na posição: {vetor.pesquisar(300)}') #retorna -1  
print(f'Na posição: {vetor.pesquisar(8)}') #retorna 2  
  
print('-' \* 30)  
vetor.imprimir()

# Vetores Ordenados

Diferente do Vetor Não Ordenado, é preciso primeiramente fazer uma pesquisa linear para saber em qual posição o dado será registrado.

E depois que essa pesquisa for feita, aí será necessário a inclusão e o remanejamento dos outros dados que ocupavam a posições, segue o princípio contrário da exclusão de vetor não ordenado.





De qualquer forma, a notação Big-O sempre será O(n), porque será necessário pesquisa e remanejamento.

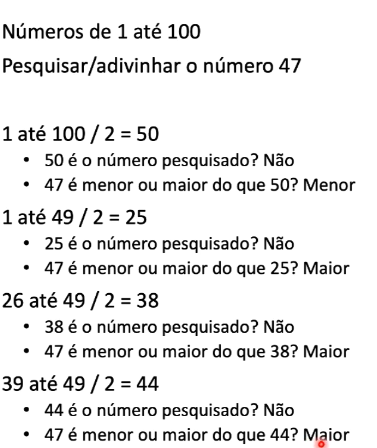
A Pesquisa tem uma diferença sutil, uma vez que o vetor está ordenado, caso procurássemos o número 10, e a posição já está no 20, o vetor já para a pesquisa porque é impossível que o número 10 esteja localizado posteriormente ao número 20. No pior caso, se o número não estiver no vetor ou estiver na última posição, terá que percorrer todas as posições do vetor. A notação big-o da pesquisa também é O(n).

<https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>

Código

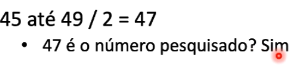
*import* numpy *as* np  
  
*class* VetoresOrdenados:  
 *def \_\_init\_\_*(self, *capacidade*):  
 self.capacidade = *capacidade* self.ultima\_posicao = -1  
 self.valores = np.empty(self.capacidade, dtype=int)  
  
 *def* imprime(self):  
 *if* self.ultima\_posicao == -1:  
 print(f'O vetor está vazio')  
 *else*:  
 *for* i *in* range(self.ultima\_posicao + 1):  
 print(f'Na posição [{i}]', '=', self.valores[i])  
  
  
 *def* inserir(self, *valor*):  
 *if* self.ultima\_posicao == self.capacidade - 1:  
 print('Capacidade Máxima atingida.')  
 *return* posicao = 0  
 *for* i *in* range(self.ultima\_posicao + 1):  
 posicao = i  
 *if* self.valores[i] > *valor*:  
 *break  
 if* i == self.ultima\_posicao:  
 posicao = i + 1  
  
 x = self.ultima\_posicao  
 #remaneja as posições  
 *while* x >= posicao:  
 self.valores[x + 1] = self.valores[x]  
 x -= 1  
  
 self.valores[posicao] = *valor* self.ultima\_posicao += 1  
  
 *def* pesquisar(self, *valor*):  
 *for* i *in* range(self.ultima\_posicao + 1):  
 *if* self.valores[i] > *valor*:  
 *return* -1  
 *elif* self.valores[i] == *valor*:  
 *return* i  
 *elif* i == self.ultima\_posicao:  
 *return* -1  
  
 *def* excluir(self, *valor*):  
 #encontra se o valor existe:  
 posicao = self.pesquisar(*valor*)  
 *if* posicao == -1:  
 *return* -1 #não existe  
 #apaga se encontrar:  
 *else*:  
 *for* i *in* range(posicao, self.ultima\_posicao):  
 self.valores[i] = self.valores[i + 1]  
  
 self.ultima\_posicao -= 1  
  
  
vetor = VetoresOrdenados(10)  
  
vetor.inserir(2)  
vetor.inserir(4)  
vetor.inserir(3)  
vetor.excluir(2)  
vetor.imprime()  
  
print(f'Posição: {vetor.pesquisar(5)}')  
print(f'Posição: {vetor.pesquisar(3)}')  
print(f'Posição: {vetor.pesquisar(4)}')  
print(f'Posição: {vetor.pesquisar(200)}')

# Pesquisa Binária

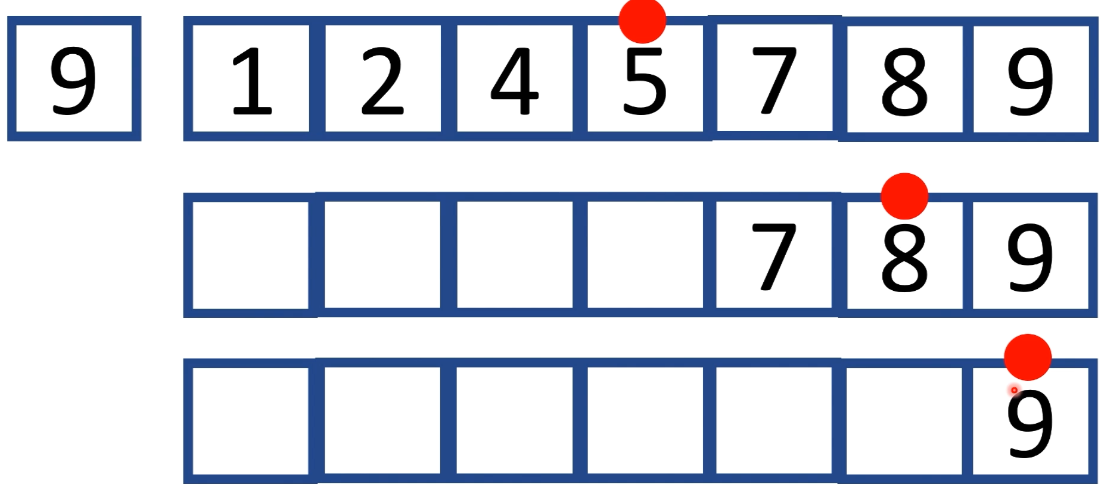


Reduz sempre a quantidade pelo intervalo, por exemplo, quando chega no passo de 26 até 49, precisamos somar (26 + 49 = 75 / 2 = 37,5 == 38).

A mesma coisa no 39 até o 49 (39 + 49 = 88 / 2 = )44



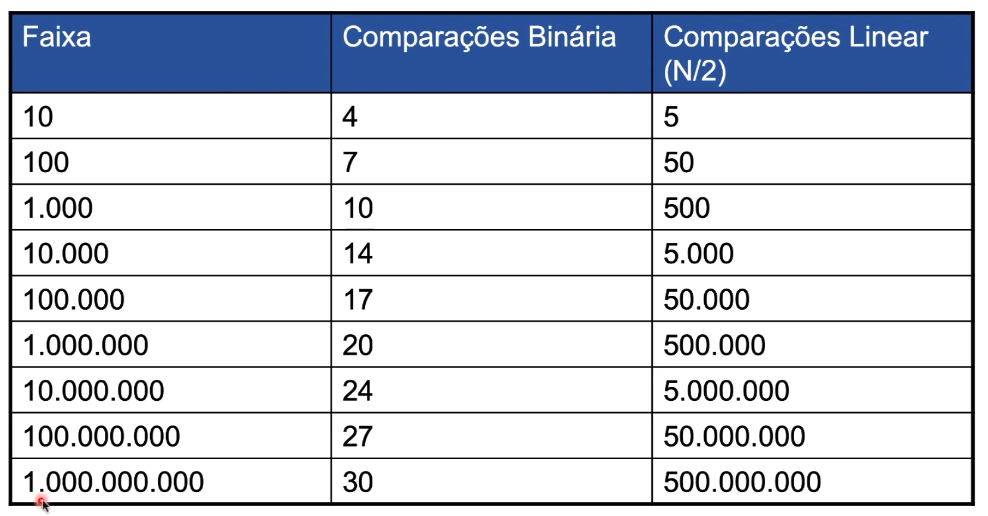
Foram necessários somente 5 passos para fazer a pesquisa. Se fosse através da PESQUISA LINEAR em um VETOR ORDENADO, levaríamos 47 passos.



Ou seja, ele verifica o primeiro, o último e encontra o do meio, e verifica se é igual, menor ou maior e repete o processo. Sempre somando e dividindo por dois.

## Tabela: Pesquisa Binária x Pesquisa Linear

Analisando a tabela, em uma faixa de 10 elementos, em uma pesquisa binária o número máximo será de 4 passos, enquanto na linear, o número máximo será de 5 no pior cenário será dos 10.



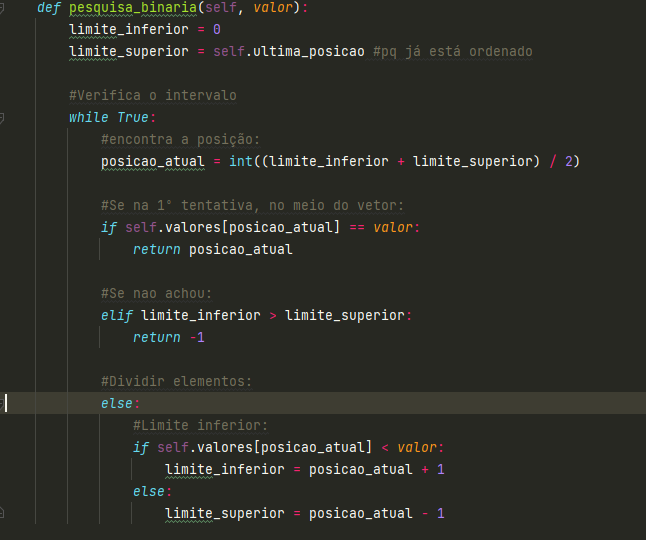
### Quando usar um ou outro

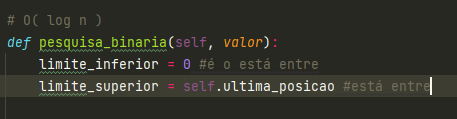
Quando não possuímos vetores ordenados, a pesquisa obrigatoriamente será a Linear, mas, se os vetores estiverem ordenados, a pesquisa binária pode e deve ser aplicada.

## Pesquisa Binária e seu Big-O

A função de pesquisa binária recebe um O(log n) é uma pesquisa que é de um ‘tipo’ quase Constante. Isso reduz muito o tempo de processamento/ Run Time.

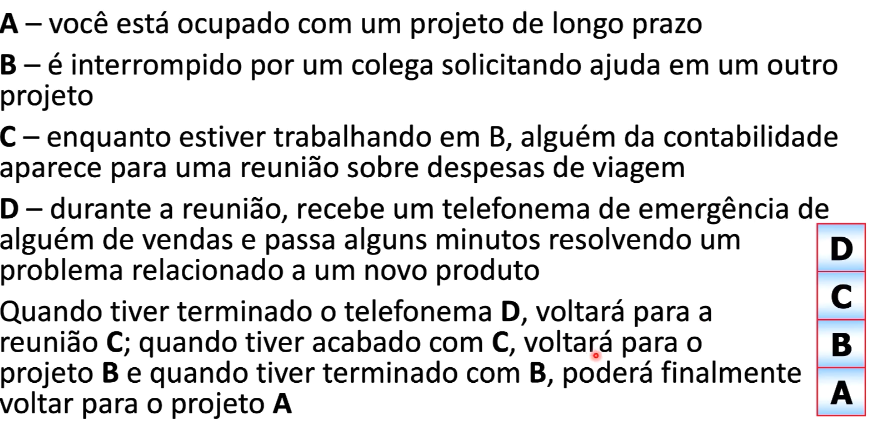
## Implementação da Pesquisa Binária





# Pilhas

Sempre que estiver fazendo algo, e algo novo acontece, entre em cima, e por aí vai, mas somente pode acessar uma, se tirar a pilha de cima.



* Permite acesso sempre e somente ao último item inserido.
* Se o último item for resolvido ou melhor, removido, então, o anterior, poderá ser acessado.

## Operações das Pilhas

**Empilhar**: Colocar um item somente no topo da pilha

**Desempilhar**: Remover um item do topo da pilha

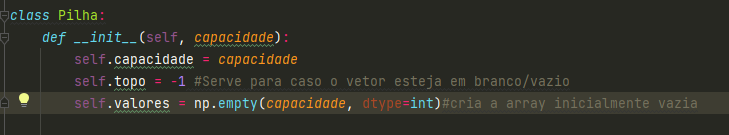
**Visualizar**: Somente o item que está no topo da pilha

## LIFO

É o ‘Last-In-First-Out’ ou , ‘Último que entra, primeiro que sai’.

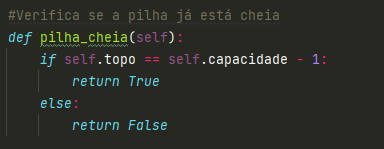
Diferente dos Vetores, que podemos pesquisas os valores conforme, nas Pilhas, somente podemos acessar o topo.

## Definindo a Class Pilha

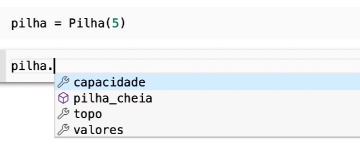


## Definindo Funções

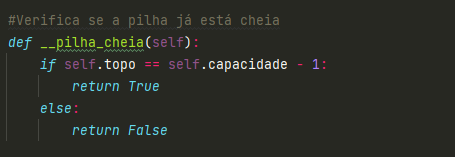
Criação de uma função específica para a classe, que o usuário não deverá ter aceso e chamar essa função.



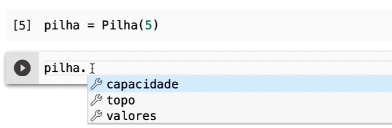
Repare que essa função pode ser acessada pelo usuário.



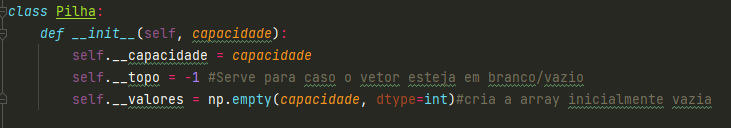
Para impedir que o usuário acesse a função, indica que é um método PRIVADO que será acessado somente dentro da classe.

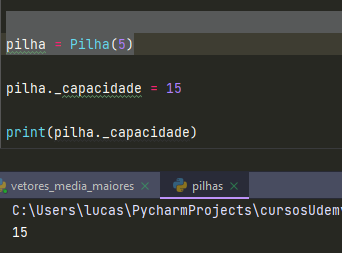


Veja que dessa forma, o usuário não consegue acessar a função que antes ele conseguia.



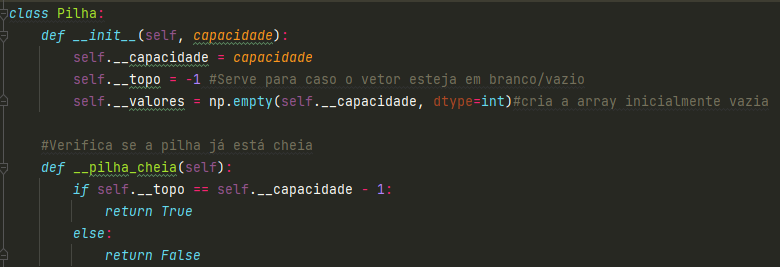
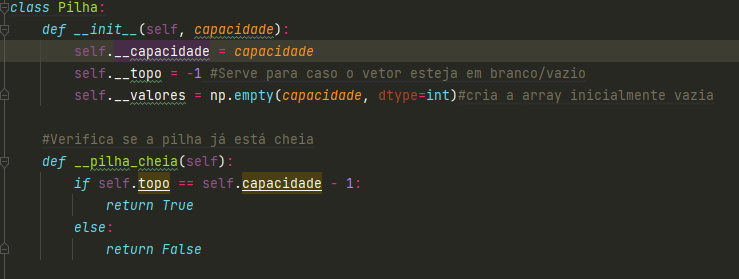
## Segurança, Atributo Privado nas Pilhas

Na verdade, todos os atributos nesse caso da pilha, devem ser alocados como PRIVADOS, por uma questão de segurança.Dessa forma, o usuário jamais poderá acessar esses atributos individualmente e assim alterar manualmente. Somente a Classe que é a portadora poderá.

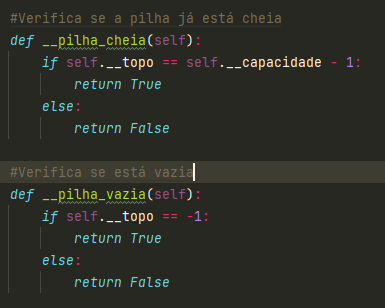
Se não fizermos essa proteção, veja que o usuário poderá alterar os atributos depois de instanciar os objetos.  
  
A pilha que deveria ter somente 5 elementos, agora tem 15, isso porque o usuário alterou.

## Atenção com Atributos Privados

Quando iniciarmos os atributos de forma privada, todas as suas chamadas nas funções devem acompanhar os “\_ \_ “ , caso contrário, terá erro!

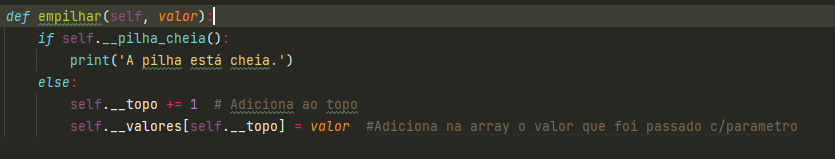


## Pilha Vazia e Cheia



## Funções – pt2

### Empilhando um item na Pilha



Veja que o único método por enquanto pode ser acionado pelo usuário é o método Empilhar. E somente o método, os atributos que compõe o método não poderão.

