

1. Introdução ao Python - IP

Resumo

| | |
|---|----------|
| Resumo | 1 |
| O que faremos essa semana | 2 |
| Para seguir o guia na sua máquina você precisa acessar: | 2 |
| O que é Python? | 2 |
| Mão na massa. | 3 |
| Como começar a trabalhar no Colab ? | 3 |
| | 6 |

O que será abordado neste mini curso?

Serão abordadas os conteúdos de introdução ao Python, a definição de Python, e os conteúdos primordiais da linguagem.

Para seguir o guia na sua máquina você só precisa acessar:

❑ **Colab** (<http://colab.research.google.com>) ou lápis e papel.

Neste guia foi utilizado o Colab.

O que é Python ?

Python é uma linguagem de programação de alto nível de abstração (comandos mais próximos da linguagem humana, do que a linguagem de máquina).



Python é uma linguagem dinâmica, interpretada, robusta, multi plataforma, multi-paradigma (orientação a objetos, funcional, refletiva e imperativa.) Foi lançada em 1991 por Guido van Rossum, é uma linguagem livre (até para projetos comerciais), e hoje pode-se utilizá-la para programar desktops, web e mobile.

Mão na Massa

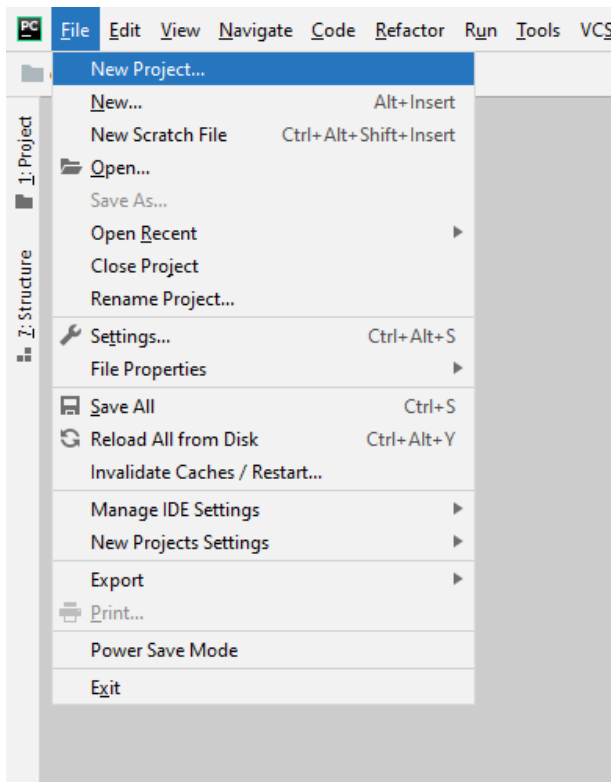
Criando seu Primeiro Projeto no Colab

Antes de Qualquer coisa, você precisa ter o PyCharm instalado na sua máquina, a instalação é simples e intuitiva, instale-o na sua máquina e vamos começar.

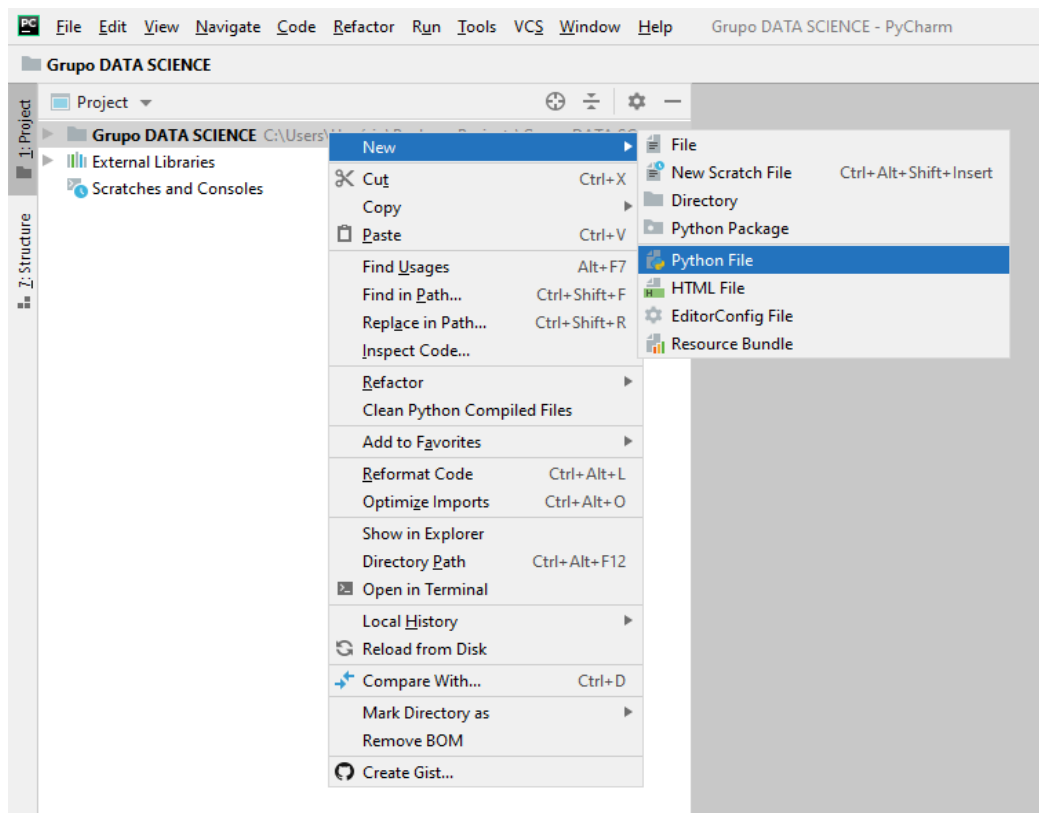
1. Como Começar a Trabalhar no Colab?

Após abrir seu PyCharm, para criar seu primeiro Projeto*, siga os passos listados abaixo:

1 - No canto superior esquerdo temos o **botão File**, clique nele e logo em seguida **clique em New project**, o PyCharm irá criar um novo Projeto, onde irá abrigar os arquivos contendo seus códigos.

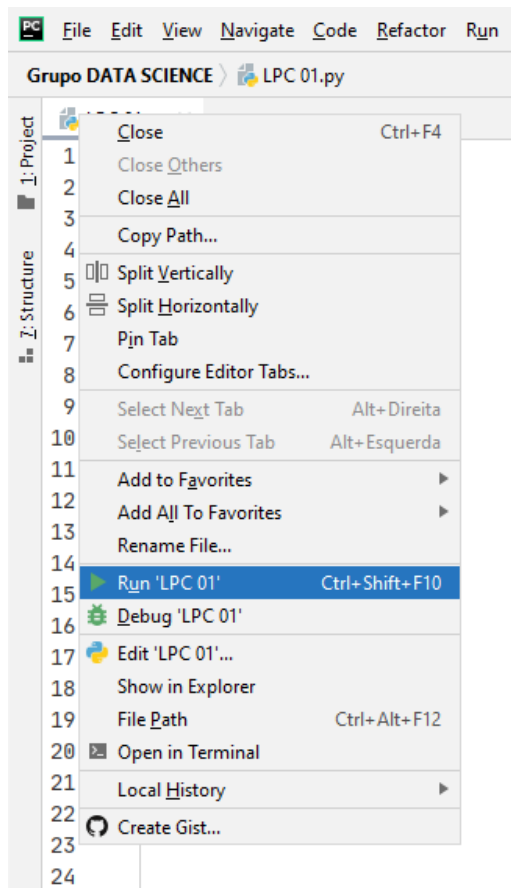


2 - Com o Projeto criado, clique em Project - Grupo DATA SCIENCE (Ou qualquer outro nome que você tenha dado ao projeto) - New - Python File e nomeie o arquivo.



Não esqueça de manter o PyCharm sempre atualizado e organizado, criando arquivos em ordem numérica, preferencialmente. Agora você já está pronto para escrever sua primeira linha de código, oficialmente, em Python.

Para executar o seus códigos, clique com o botão direito no seu arquivo, no canto superior direito, logo abaixo do nome do projeto (Grupo DATA SCIENCE) e clique em - Run 'Nome do Arquivo'



2. Tipos Primitivos de Dados em Python.

No Python temos 4 tipos de dados, **Inteiros**(int), **Reais**(float), **Booleanos** - Verdadeiro/Falso (bool - True/False), e **Caracteres** (string).

| | |
|---|-----------------------------|
| 1 | <code>x = 2</code> |
| 2 | <code>print(type(x))</code> |

Neste exemplo, atribuímos o valor de 2 a variável x e pedimos ao programa para que escreva na tela, usando o comando **print*** o tipo de dado da variável x.

O programa nos retorna o seguinte resultado : “<class 'int'>”, ou seja, a classe, ou o tipo de dado da variável x é inteiro (int).

- Comando *print*: **digite print*** e, entre parênteses, você escreve o que o programa deve imprimir na tela, no caso do exemplo, o tipo de dado da variável x*, veremos com detalhes mais a frente.

- Comando *type*: **digite type* e, entre parênteses, você escreve uma variável para saber o seu tipo, no caso do exemplo , está sendo requisitado o tipo de dado da variável x, e ele retorna int, ou inteiro.**

```
1 x = 'Olá mundo!'
2 print(x)
3 print(type(x))
```

Neste segundo exemplo, nós atribuímos à variável x, a string “Olá mundo”. Para escrever uma string em python, uma **cadeia de caracteres**, a string sempre deve estar dentro de aspas, seja elas simples ou duplas, o recomendado é utilizar aspas simples.

Como resultado, o programa nos retorna o seguinte: **Olá mundo! , <class 'str'>.**

O programa imprimiu a string atribuída a x (Olá mundo!), na tela e nos retornou a classe dela, (str, ou string).


```
1 x = 2
2 y = 3
3 print(x == y)
```

Dados booleanos são um pouco mais complexos.

Entendendo o código acima:

Atribuímos à variável x o valor 2. atribuímos a y o valor de 3, e pedimos pra imprimir na tela “x == y”, ou seja, x igual a y. se x for igual a y ele imprime True, se x for diferente de y ele imprime False, como 2 é diferente de 3, ele imprime False. (dois sinais de igual juntos simbolizam igualdade, um único sinal de igual é utilizado somente em atribuição de valores, vamos entender isso melhor mais pra frente)

Dados booleanos são extremamente úteis e serão usados ao longo de toda sua vida como programador(a).



Dados do tipo float são feitos para trabalhar com números quebrados, com casas decimais, e com números negativos. a lógica é a mesma da dos números inteiros.

3. Operadores Aritméticos em Python

O Python pode ser utilizado como uma calculadora matemática avançada. Praticamente, todos os operadores aritméticos funcionam da mesma forma como os conhecemos da matemática elementar. Por exemplo, para trabalharmos com as **4 principais funções matemáticas**, a soma, subtração, multiplicação e divisão, temos os operadores conforme tabela a seguir.

| Operação | Operador |
|---------------|----------|
| adição | + |
| subtração | - |
| multiplicação | * |
| divisão | / |

Temos também, operadores para exponenciação, divisão inteira, e resto de divisão.

| Operação | Operador |
|---------------|----------|
| exponenciação | ** |

| | |
|------------------|----|
| divisão inteira | // |
| resto da divisão | % |

```

1 x = 2**3
2 print(x)

```

A exponenciação é feita da seguinte forma : base**expoente, no exemplo, temos 2**3, ou seja, 2 elevado a 3ª potência; o programa retorna o valor 8.

```

1 x = 10//3
2 print(x)

```

A divisão inteira é feita pelo operador (//) e nos retorna a parte inteira de uma divisão, por exemplo, 10 dividido por 3 ? O resultado é 3,333333.... porém o programa só nos retorna a parte inteira da divisão, ou seja, 3.

```

1 x = 10%3
2 print(x)

```

Para a operação de resto da divisão, podemos utilizar o mesmo código do exemplo anterior, abordamos a operação 10//3 == 3, essa operação tem o resto 1, na operação de resto de divisão nós pedimos ao programa que nos retorne justamente esse valor restante, sendo, 10 % 3 igual a 1.

4. Operadores Relacionais Simples.

Existem 3 relações possíveis entre 2 operandos, são elas:

| Descrição | Operador |
|-----------|----------|
| | |

| | |
|-----------|----|
| Maior que | > |
| Menor que | < |
| Igual a | == |

Para obtermos a relação entre 2 membros, temos que utilizar a seguinte estrutura:

<membro à esquerda> OPERADOR <membro à direita>

É importante observar que a inversão dos membros ocasiona na inversão do resultado da expressão, isto é, se o membro que estiver a esquerda for para a direita e vice-e-versa, a relação entre eles será o contrário.

5. Operadores Relacionais Compostos.

| Descrição | Operador |
|------------------|----------|
| Maior ou igual a | >= |
| Menor ou igual a | <= |
| Diferente de | != |

Maior ou igual a: Verifica se o valor A é maior ou igual ao valor B.

Menor ou igual a: Verifica se o valor A é menor ou igual ao valor B.

Diferente de: Verifica se o valor A é diferente do valor B.

Testem esse padrão, testando todos os operadores relacionais, prestando atenção no resultado, True ou False.

| | |
|---|-------------------------------|
| 1 | <code>a = 2</code> |
| 2 | <code>b = 3</code> |
| 3 | <code>print(a >= b)</code> |

6. Função print().

A função primordial, utilizada para simplesmente imprimir na tela, a função print() imprime variáveis, valores, strings, e todos juntos também, na sua tela. vamos para o exemplo prático para entender melhor.

| | |
|---|---|
| 1 | <code>print('Olá mundo!', 2, 2.5, True, False)</code> |
|---|---|

A função print, no exemplo, está imprimindo a string “Olá mundo!” strings sempre entre aspas, o número 2, o número 2.5, e os valores True e False.

Para imprimir diferentes tipos de dados dentro de um único print, eles têm sempre que estar separados por vírgula.

Podemos também formatar uma string para receber valores dela, para isso, simplesmente coloque a letra “f”, antes de escrevê-la, e digite o valor que deverá ser incluído na função entre chaves.

```
1 numero = 2
2 print(f'O número {numero} é par.')
```

Entendendo este código:

Atribuímos o valor 2 a variável `numero`, na linha seguinte, chamamos a função `print`, formatada (letra `f` antes da string), Lembre-se que, para incluir valores externos dentro de uma string, ela precisa estar entre chaves, como no exemplo, a variável `número` está entre chaves. O programa nos retorna o seguinte: **O número 2 é par.**, o programa SEMPRE vai “colocar” o valor que está dentro da variável no espaço formatado.

7. Função `input()`.

Nem sempre vamos trabalhar somente com valores já determinados, às vezes, precisamos pedir ao usuário que digite um valor, para aí sim, podemos trabalhar em cima dele, A função `input` serve justamente para isso, ela pede ao usuário que digite um valor, e o guarda dentro de uma variável, como no exemplo abaixo.

```
1 nome = input('Digite seu nome: ')
2 print(f'Seu nome é {nome}.')
```

***OBS: Sempre digite os exemplos no seu PyCharm, para praticar e se habituar com a linguagem.**

Bom, no exemplo acima, **nós pedimos ao usuário que digite seu nome**, e em seguida nós o imprimimos, Seu nome é {nome digitado pelo usuário}.

OBS*: A função `input` pode ser modificada de acordo com suas necessidades, podendo ser convertida para inteiro e para float(número real).

Observe que, se, por exemplo você precisar saber a idade do usuário e escrever desse mesmo jeito:

```
1 idade = input('Digite sua idade: ')
2 print(f'Você tem {idade} anos.')
```

Teste esse código, funcionou normalmente não é ?

Não, não funcionou corretamente, e eu vou te dizer o porquê no exemplo abaixo.

```
1 idade = input('Digite sua idade: ')
2 print(f'Você tem {idade} anos.')
```

```
3 print(type(idade))
```

Ao pedir a idade do usuário, um número inteiro, ele imprime tudo normalmente, mas no momento que você pede o tipo de dado da variável idade, ele nos retorna : **<class 'str'>**, “ué, mas como, se o número 23 é um número inteiro, por que ele nos retornou tipo string ? “

Veja bem, a função input(), ela vem pré setada para receber strings, lembre-se que, strings são cadeias de caracteres, sejam eles números, ou símbolos ou qualquer outra coisa.

Para receber números e tê-los como do tipo inteiro, precisamos **converter o input para inteiro**.

Para converter a função input() para inteiro, apenas digite:

```
int(input('Mensagem dizendo o que deve ser informado pelo usuário'))
```

```
1 idade = int(input('Digite sua idade: '))
2 print(f'Você tem {idade} anos.')
3 print(type(idade))
```

Fazendo essa alteração, pedindo o type, ou o tipo de dado da variável idade, o programa nos retorna: **<class 'int'>**, e assim que fizermos essa alteração, ele receberá somente números inteiros, qualquer outra coisa será tratada como erro.

Mesma coisa para valores float (reais), conforme exemplo:

```
1 peso = float(input('Digite seu peso: '))
2 print(f'Você pesa {peso}KGs')
3 print(type(peso))
```

O programa retorna: **<class 'float'>**.

8. Estruturas Condicionais e Repetições.

Muitas vezes, precisamos estabelecer certas “condições” dentro do código, se a condição for cumprida, executamos x funcionalidade, e caso não, executamos outra. estruturas condicionais servem para isso.

Se condição == verdadeira:

bloco de código para caso condição == verdadeiro

se não:

bloco de código para caso condição == falso

No python, o se e o senão são representadas, pelas funções:

if e else;

As funções, `faca` e `então`, do visualg, não existem em python, eles são substituídos pelos dois pontos “:” e pela indentação. Em Python, a indentação possui função bastante especial, até porque, os blocos de instrução são delimitados pela profundidade da indentação, isto é, os códigos que estiverem rente a margem esquerda, farão parte do primeiro nível hierárquico. Já, os códigos que estiverem a 4 espaços da margem esquerda, estarão no segundo nível hierárquico e aqueles que estiverem a 8 espaços, estarão no terceiro nível e assim por diante.

***PARA CRIAR UMA INDENTAÇÃO EM PYTHON, UTILIZE A TECLA “TAB” DO SEU TECLADO, E ELE CRIARÁ A INDENTAÇÃO CORRETA (DE QUATRO ESPAÇOS), AUTOMATICAMENTE.**

Todos os blocos são delimitados pela profundidade da indentação e por isso, a sua importância, é vital para um programa em Python. O mau uso, isto é, utilizar 4 espaçamentos enquanto deveríamos estar utilizando 8, acarretará na não execução, ou então, no mal funcionamento em geral.

Para entendermos melhor vamos utilizar um exemplo juntando todo o conhecimento adquirido até agora.

```
1  num = int(input('Digite um número: '))
2  if num % 2 == 0:
3      print(f'0 número {num} é par')
4  else:
5      print(f'0 número {num} é ímpar')
```

Perceba que os códigos `print(f'o número {num} é par')` e `print(f'o número {num} é ímpar')` estão deslocados mais a direita. eles estão no chamado **segundo nível hierárquico**, eles só serão executados caso a condição imposta seja cumprida.

Os códigos colados na margem à esquerda são os que estão no **primeiro nível hierárquico**, e serão executados sempre.

Entendendo o código:

```
1 num = int(input('Digite um número: '))
2 if num % 2 == 0:
3     print(f'O número {num} é par')
4 else:
5     print(f'O número {num} é ímpar')
```

Primeiro, pedimos ao usuário que digite um número, e depois a seguinte lógica é feita:

Se o resto da divisão do número por 2 for igual a zero:

imprime(o número {num} é par)

se não:

imprime(o número {num} é ímpar)

Todos concordamos que um número é ímpar, quando o resto da sua divisão por 2 é 0, ou seja é uma divisão exata. Caso tenha algum resto ele é considerado ímpar.

9. Estruturas Condicionais Aninhadas e o elif;

Para estabelecer mais de uma condição, precisamos utilizar o comando **elif**, que pode ser utilizado múltiplas vezes para estabelecer múltiplas condições diferentes:

```
1 num = float(input('Digite um número: '))
2 if num > 0:
3     print('Este número é positivo')
4 elif num == 0:
5     print('Este número é neutro')
6 else:
7     print('Este número é negativo')
```

Perceba que foi adicionada mais uma condição, para caso o número seja igual a zero, isso pode ser feito diversas vezes, as estruturas condicionais sempre seguem esse padrão **if - else, ou if - elif - else**; nunca se deve começar com else, ou com elif, sempre siga esse padrão.

Ao final de cada condição, use os dois pontos “:”, enter para avançar para a próxima linha, e automaticamente, criar a indentação.

10. Operadores Lógicos: and; or; not; in; not in e a Função range().

Operador and:

O “e” é o operador and, em Python. Seja o seguinte teste:

teste condição1 and condição2:

O programa só retornará **True** se as DUAS condições forem cumpridas, caso somente uma seja cumprida, ele retornará **False**

```
1 resposta=int( input('Qual sua idade: ') )
2 if resposta>=18 and resposta <=65:
3     print('Você é obrigado a votar!')
4 else:
5     print('Você não é obrigado a votar!')
```

Veja que para o IF ter resultado verdadeiro, ambas as condições devem ser verdadeiras: tanto deve ter 18 anos ou mais E deve ter 65 anos ou menos, Essa é a característica do operador and.

Todos os testes devem ser true, para o resultado ser true.

Se um deles for false, o resultado é falso.

Operador or:

O operador or é parecido, mas completamente diferente do operador and, ele usa a mesma forma de digitação ao ser aplicado.

O or trabalha com diversas possibilidades, por exemplo, para executar um programa ele precisa que as seguintes condições sejam cumpridas(usaremos letras do alfabeto para representar as condições): **a OU b OU c OU d**: no caso do **and** ele precisa que todos sejam verdade, no operador **or** ele precisa que somente uma das condições seja verdade para que ele retorne verdade. Ficou meio confuso ? Veja o exemplo:

```
1 print('1. Idoso')
2 print('2. Gestante')
3 print('3. Cadeirante')
4 print('4. Atendimento comum')
5 resposta=int( input('Selecione uma opção: ') )
6
7 if (resposta==1) or (resposta==2) or (resposta==3):
8     print('Você tem direito a fila prioritária')
9 else:
10    print('Aguarde sua vez na fila comum.')
```

Se qualquer um deles for verdade, esse **if** vai ser verdade, pois usamos o operador lógico or (OU). Pra ser prioridade, OU você tem que ser idoso, OU tem que ser gestante OU tem que ser cadeirante.

Não precisa ser os três, basta um deles ser verdade, que o teste retorna verdade.

***OBS: Pratique, a prática é primordial para entender os conceitos, apenas ler sobre não fará você entender todos os assuntos de primeira, se for o caso, digite o código exemplo no seu PyCharm apenas para ver como ele funciona **NÃO COPIE E COLE**.**

Operador not:

O operador not é o mais simples de todos, ele pega uma expressão e a inverte, caso ela seja verdade, ela será falsa e vice versa

```
1 a = 6
2 b = 3
3 print(not a > b)
```

Como sabemos, 4 é maior que 2, então a expressão deveria retornar **True**, Porém como usamos o **not** antes da expressão ele inverteu o seu resultado. teremos **False** como resultado.

um exemplo mais complexo utilizando **not**;

```
1 agencia = input('Qual é a sigla da agência espacial norte-americana?')
2
3 if not agencia=='NASA':
4     print('Errado, tente novamente!')
5 else:
6     print('Correto, NASA!')
```

Função range():

Como funciona a função range do Python? A função "range()" retorna uma lista no intervalo definido. Assim, temos uma forma rápida e fácil para gerarmos listas de valores numéricos que estejam num determinado intervalo, por exemplo,

range(inicio, fim, passo)

por exemplo, **range(1, 100, 2)**, ele cria uma **lista** todos os números num intervalo de 1 até 100 pulando de 2 em 2, (1, 3, 5, 7...) Caso queira um passo normal, escreva somente o início e o passo

range(1, 7): representa todos os números num intervalo de 1 até 6, “ué, não é entre 1 e 7 ? tem o 7 ali ó”, **não**, é uma regrinha da sintaxe do Python, ele ignora o último número, e só representa até o penúltimo.

o range é extremamente útil naqueles momentos, você tem que ter todos os números de 1 até 1000 na tela, você vai fazer, imprimir um por um?, logicamente não, você utilizará a função range, ela é muito utilizada em conjunto com o loop for, que veremos em outro momento.

Operador **in**; not in:

Lembra da função range() ? ela será também importante para esses dois aqui, os operadores “in” e “not in” representam exatamente “dentro de” “não está dentro de”, são usados com a função range(), por exemplo:

```
1 c = int(input('Digite um número: '))
2 if c in range(1, 100, 2):
3     print(f'{c} está entre 1 e 99.')
4 else:
5     print(f'{c} não está entre 1 e 99.')
```

“a recebe 3, se a estiver num intervalo entre 1 e 300: escreva : “a está entre 1 e 300”, se não escreva “Não está entre 1 e 300””, como a é 3, e 3 está entre 1 e 300 ele retorna True, e imprime o bloco dentro do if.

O not in é exatamente o Inverso, teste o código acima, substituindo o **in** por **not in** e veja o resultado.

A lógica, substituindo o **in** por **not in** é:

“a recebe 3, se a **não estiver** num intervalo entre 1 e 300: escreva : “a está entre 1 e 300”, se não escreva “false””, **como a está num intervalo entre 1 e 300**, ele retornará false e escreverá o bloco dentro do else.

Exercícios e Desafios

1. Faça um Programa que mostre a mensagem "Olá mundo" na tela.
2. Faça um Programa que peça um número e então mostre a mensagem O número informado foi [número].
3. Faça um Programa que peça dois números e imprima a soma.
4. Faça um Programa que peça as 4 notas bimestrais e mostre a média.
5. Faça um Programa que converta metros para centímetros.
6. Faça um Programa que peça o raio de um círculo, calcule e mostre sua área.
7. Faça um Programa que calcule a área de um quadrado, em seguida mostre o dobro desta área para o usuário.
8. Faça um Programa que pergunte quanto você ganha por hora e o número de horas trabalhadas no mês. Calcule e mostre o total do seu salário no referido mês.
9. Faça um Programa que peça um valor e mostre na tela se o valor é positivo ou negativo.
10. Faça um Programa que verifique se uma letra digitada é "F" ou "M". Conforme a letra escrever: F - Feminino, M - Masculino, Sexo Inválido.
11. Faça um Programa que verifique se uma letra digitada é vogal ou consoante.

Obrigado! **Thank you!**
Dúvidas?

Conheça: <https://www.clovesrocha.digital>
`print("CONHEÇA MEU CANAL");`



m.youtube.com/ConsultorCloves

