Escrito por: Artur de Oliveira Barbosa
GitHub: https://github.com/Space2544
Email: arturoliveira886@gmail.com

"conhecimento é poder" -Francis Bacon

==========

# Documentação Python

Disclaimer:

Esta será uma documentação/anotação sobre o Python, que na minha opinião é uma das linguagens de programação mais interessantes que tem por ai. É uma junção de cursos, documentações, fóruns, livros, vídeos que assisti, que eu fiz uma leitura e resolvi anotar pra formar essa documentação.

## Uma breve história sobre o Python

**Python** é uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991.

Atualmente, possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem, como um todo, não é formalmente especificada. O padrão de facto é a implementação CPython.

A linguagem foi projetada com a filosofia de enfatizar a importância do esforço do programador sobre o esforço computacional. Prioriza a legibilidade do código sobre a velocidade ou expressividade. Combina uma sintaxe concisa e clara com os recursos poderosos de sua biblioteca padrão e por módulos e frameworks desenvolvidos por terceiros.

Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural.

Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens.

Devido às suas características, ela é utilizada, principalmente, para processamento de textos, dados científicos e criação de CGI's para páginas dinâmicas para a web.

Foi considerada pelo público a 3ª linguagem "mais amada", de acordo com uma pesquisa conduzida pelo site Stack Overflow em 2018 e está entre as 5 linguagens mais populares, de acordo com uma pesquisa conduzida pela RedMonk.

O nome Python teve a sua origem no grupo humorístico britânico Monty Python, criador do programa Monty Python's Flying Circus, embora muitas pessoas façam associação com o réptil do mesmo nome (em português, píton ou pitão). (**wikipédia**).

==========

Como foi possível ver, a historia das linguagens de programação sempre envolveram muitas coisas e situações que aconteceram e que foram feitas pra resolver determinado problema e claro isso não foi diferente com o **Python**.

O bom do Python é que essa é uma das linguagens que podem ser usadas pra mais de uma única coisa, desde data science, machine learning até mesmo criação de apps e jogos.

==========

Agora que sabemos um pouco mais sobre a história dessa linguagem vamos para o que realmente interessa por assim dizer.

==========

## **Comandos Básicos**

**Print ()**: Serve para escrever um texto, comentário dentro de parênteses, mas se quiser escrever algo será necessário aspas.

Exemplo:

```
>>> print ('hello, world!')
hello, world!
```

==========

**=** : O simbolo de "igual" em Python significa "recebe". Ou seja um **variável** vai receber tal **valor**.

==========

**,** : O simbolo da “virgula” em Python tem a função de **juntar** letras,  números e etc.
Ou simplesmente fazer uma “**pausa**” se caso você quer que duas variáveis apareçam na mensagem.

Exemplos:

```
>>> x = 1
>>> y = 2
>>> print(x,y)
1 2
```

```
x = input('coloque um numero: ')
y = input('coloque um outro numero: ')
print('a junção desses dois numeros é:', x + y)
```

==========

**Input()**: O comando **input** serve para você colocar uma “resposta” na sua **variável**.

Exemplo:

```
x = input('coloque um numero: ')
y = input('coloque um outro numero: ')
print('a junção desses dois numeros é:', x + y)
```

```
coloque um numero: 12
coloque um outro numero: 08
a junção desses dois numeros é: 1208
```

==========

## Tipos Primitivos

Em python temos vários comandos que são denominados **tipos primitivos**, cada um desses comandos fazem algo em relação a uma determinada função que você quer que o computador execute, ou melhor dizendo o seu script e etc.
Citarei 4 desses **Tipos primitivos** e o que cada um faz:

**Int()**: É uma função que converte todo o valor colocado dentro de ()
em um número inteiro, ou seja se você digitar algum número de ponto flutuante ou melhor dizendo “quebrado” como por exemplo o pi (3.1415...) vai dar erro pelo fato de não ser um número **int**eiro.

Exemplo:

```
>>> n1 = int(input('digite um número ai: '))
digite um número ai: >? 0.1
Traceback (most recent call last):
  File "<input>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '0.1'
```

==========

**Float()**: Transforma todo o valor colocado dentro em um **ponto flutuante** ou **número** “**quebrado**”.

Exemplo:

```
>>> n1 = float(input('coloque um numero quebrado: '))
coloque um numero quebrado: >? 20.32
```

E claro somente números “quebrados” se não ocorrerá o erro também.

==========

**Bool()**: Uma função que pode dizer pra você se determinado valor é **True** ou **False**, ou seja se há algum valor **Verdadeiro** ou **Falso** essa função vai te dizer isso. Desde se um valor y é maior que x, ou se o valor x é maior que y enfim, o **Bool()** serve pra basicamente isso mostrar valores **verdadeiros** ou **falsos**.

Exemplos:

```
x = bool(input('coloque um número: '))
print('o valor é {}'.format(bool (x)))
```

```
coloque um número: 1
o valor é True
```

```
coloque um número:
o valor é False
```

==========

**Str()**: Também chamado de **String**, é uma função que converte um valor para uma **string**, então você consegue até mesmo combinar uma **string** com outra.

Exemplo:

```
>>> x = str(input('coloque um numero: '))
coloque um numero: >? 01
>>> y = str(input('coloque uma letra: '))
coloque uma letra: >? ab
>>> print(x, y)
01 ab
>>> print(x,y)
01 ab
>>> print(x + y)
01ab
```

==========

**Type()**: Serve para verificar qual é o tipo primitivo que está sendo utilizado na linha comando que você escreveu. (Print também é uma string).

Exemplo:

```
>>> x = str(input('olá como você vai? '))
olá como você vai? >? Muito Bem
>>> print(type(x))
<class 'str'>
```

==========

## Funções Embutidas

O interpretador do Python possui várias funções e tipos embutidos que sempre estão disponíveis. Essas funções dependendo do que você quer fazer no seu codigo você pode por exemplo deixar mais organizado e com menos linhas de codigo no seu script.

Aqui vai uma lista das funções embutidas:

**Pow()**: essa função matemática que temos em python é um acrônimo para **Power** que em português seria **potência**, isso é uma forma alternativa de fazer calculos que envolvem potência.

Exemplo:

```
>>> pow(5, 5)
3125
```

Nesse caso está 5 elevado a 5 ou (5.5.5.5.5).

**Format()**: O **format** como o próprio nome já diz, essa função **formata** um valor e faz uma representação formatada, dependendo do codigo que você está escrevendo você consegue até mesmo deixar um script que daria 5 linhas de comando, usando o **Format** você consegue diminuir até 2 linhas ou até mesmo pra 1 linha só.

Exemplos:

```
x = int(input('Por favor coloque o primeiro número a ser calculado: '))
y = int(input('Coloque outro número a ser calculado: '))
print('A soma entre esses dois numeros é: {}\nA subtração desses dois numeros é: {}'.format((x+y), (x-y)))
print('A divisão entre esses dois numeros é: {}\nA multiplicação entre esses dois numeros é: {}'.format((x/y), (x*y)))
print('A potencia entre {} e {} é: {}\nA raiz quadrada de {} é: {}'.format(x, y, (x**y), x,  (x**(1/2))))
```

```
Por favor coloque o primeiro número a ser calculado: 100
Coloque outro número a ser calculado: 2
A soma entre esses dois numeros é: 102
A subtração desses dois numeros é: 98
A divisão entre esses dois numeros é: 50.0
A multiplicação entre esses dois numeros é: 200
A potencia entre 100 e 2 é: 10000
A raiz quadrada de 100 é: 10.0
A raiz quadrada de 2 é: 1.4142135623730951
```

Nessas duas imagens eu fiz uma calculadora com apenas 6 linhas de codigos usando a formatação, e claro sempre que formatar tem que  usar chaves {}, por que assim o intepretador colocará as variaveis dentro de {}.

**\n**: O **\n** serve para continuar o código em uma outra linha na hora de executar o script, sendo assim você não precisa gastar tanto na hora de escrever o script.

==========

## Tipos Embutidos

No python temos os **tipos embutidos** e normalmente os principais **tipos embutidos** são númericos, sequências, mapeamentos, classes, instacias e exceções.

Lista de comandos do que são e o que fazem:

**.isalpha()**: esse comando serve para dizer se a resposta nesse caso é totalmente alfabética.

Exemplo:

```
x = str(input('verificador de letras e números, coloque uma letra, numero: '))
print('esse número/letra é alfabético {} '.format(x.isalpha()))
```

```
verificador de letras e números, coloque uma letra, numero: a
esse número/letra é alfabético True
```

==========

**.isupper()**: esse comando diz se está tudo em maiúsculo.

**.islower()**: esse comando diz se está minusculo.

**.istitle()**: esse comando diz se o inicio está em maiúsculo.

**.isalnum()**: esse comando diz se é alfa-númerico.

==========

## Operadores Matemáticos ou Aritiméticos

Os **operadores matemáticos** ou **aritiméticos** em python tem algumas peculiaridades a mais do que se teria quando vamos fazer algum calculo matemático, como de **adição**, **subtração**, **multiplicação**, **divisão**, **exponenciação** e etc.
Agora quando se trata de programação a duas vantagens bem claras: a primeira é que você que manda e o computador resolve, e a segunda é que você não quebra a cabeça quando ordenar o computador a fazer isso.

Aqui vai uma lista dos comandos dos **operadores matemáticos**:

**+**: adição

**-**: subtração

*: multiplicação

**/**: divisão

**: exponenciação

**//**: divisão inteira

**%**: resto da divisão

****(1/2)**: raiz quadrada

****(1/3)**: raiz cubica

Em algumas “screenshots” atrás que eu tirei, eu mostrei como se utiliza o .**format**, \ **n** juntamente com os **operadores aritiméticos**.

```
x = int(input('Por favor coloque o primeiro número a ser calculado: '))
y = int(input('Coloque outro número a ser calculado: '))
print('A soma entre esses dois numeros é: {}\nA subtração desses dois numeros é: {}'.format((x+y), (x-y)))
print('A divisão entre esses dois numeros é: {}\nA multiplicação entre esses dois numeros é: {}'.format((x/y), (x*y)))
print('A potencia entre {} e {} é: {}\nA raiz quadrada de {} é: {}'.format(x, y, (x**y), x,  (x**(1/2))))
print('A raiz quadrada de {} é: {}'.format(y, (y**(1/2))))
```

Então para frizarmos melhor estou mostrando de novo, para que fique claro o processo, muito provavelmente você se questionou o fato de ter tantos **()** e isso se deve por conta de fecharmos a linha de comando, que de inicio começou com apenas um parênteses e terminou com com esse tanto:

```
.format((x+y), (x-y)))
```

E claro que temos algo a mais também como por exemplo, se você quiser repetir algo umas 10x ou 15x sem precisar digitar e fazer com que o computador faça isso pra você, bem como foi possível perceber você vai utilizar o simbolo de *:

Exemplo:

```
print('>'*10)
>>>>>>>>>>
```

O que tiver entre ‘’ você pode colocar o que quiser desde letras,  números, o que for.

==========

## Ordem de Precedência

Outra coisa que o python tem que temos também em matematica é a **ordem de precedência**, ou seja o que vai ser calculado primeiro, pra isso temos uma ordem em que o python segue que nem em matemática, mas claro em python temos algumas coisas a mais em relação a isso.
Tá mas é realmente útil essa ordem?, Sim. Muito útil mesmo já que se você analisar direito uma resposta dependendo você pode ter errado alguma coisa e parecer que foi tudo certo e como esperado.
Bem depois dessa introdução a **ordem de precedência** vamos as ordem de cada um.

O python vai fazer o calculo:

**Primeiro**: **()**

**Segundo**: **

**Terceiro**: *, /, //, **%.**

**Quarto**: **+**, **-**

mas lembrando, se um comando que apareceu por mais que seja + ou **, se apareceu somente um desses, ai o computador irá resolver primeiro o que apareceu nesse caso:

```python
print('oi, temos um calculo a resolver, e a expressão é x ao quadrado menos y onde você pode colocar quem é x e y: ')
x = int(input('coloque um numero (X): '))
y = int(input('coloque outro numero (y): '))
print('o resultado de {} ao quadrado menos {} é: {}'.format(x, y, (x**2 - y)))
```

```
oi, temos um calculo a resolver, e a expressão é x ao quadrado menos y onde você pode colocar quem é x e y:
coloque um numero (X): 10
coloque outro numero (y): 20
o resultado de 10 ao quadrado menos 20 é: 80
```

==========

## Módulos do Python

Os **módulos do python** fazem parte de uma biblioteca que o python tem, tanto que pra instalar algum módulo é necessário acessar o terminal do python para você baixar um desses modulos pelo seguinte comando: **pip install** (nome do modúlo).

Mas claro, isso varia de sistema para sistema, ou seja o comando pode mudar, pode ser que seja assim: **pip3 install** (nome do módulo).

Basicamente o python consegue fazer muita coisa a mais com módulos que você vai e baixa, logo o que dá pra fazer com os módulos é muito vasto, claro que é necessário você pesquisar muito sobre o que cada módulo faz, se existe uma documentação pra isso ou alguma wiki sobre enfim pesquise antes de aplicar.
Já que cada um tem uma ação muito especifica, por exemplo você pode instalar um módulo para criar **bots** no **Discord** utilizando o comando ***pip install discord***.

Outro exemplo, é que você pode utilizar o rich, que tem funções básicas mas muito completas e pode ajudar a criar programas mais bonitos visualmente e organizados. Ele pode ser instalado com o comando: **pip install rich**, e importado no arquivo .py da seguinte forma: **import rich**.

Isso foi um pequeno exemplo do que é possível se fazer com esse módulo, agora se você achar um que seja útil até mesmo fora do python você até pode melhorar o desempenho do seu pc, notebook, mas claro isso só seria uma informação a mais, não são todos os módulos do python que fazem isso.

Para poder acessar os módulos que baixou ou que já estão baixados será necessário dar **import** + (nome do módulo).

Quando você der esse comando você está selecionando todo o módulo, ou seja tudo o que tiver dentro do módulo.

Mas suponhamos que você queira somente algo mais especifico do modulo em si, pra isso você precisará dar o comando **from** (nome do módulo) **import** (nome de uma parte determinada do módulo).

Um módulo que já vem por padrão no Python por exemplo é o **math**, o módulo math como o nome já diz é um modulo matemático, logo dentro desse módulo temos varias ferramentas, irei citar algumas abaixo:

**Ceil**: serve para arredondar um número positivamente, ou seja 8.02 vai virar 9.

**Floor**: serve para arredondar o número negativamente, ou seja 9.82
vai virar apenas 9.

**Trunc**: serve para eliminar (**truncar**) o que há atrás de um número que tem ponto. Exemplo 12.3124 >>> 12.

**Pow**: serve para calcular um número elevado (power, potência) a algum outro número, por exemplo ao quadrado ou ao cubo.

**Sqrt**: serve para mostrar a raiz quadrada, cubica ou qualquer outro valor desse número (**Sq**uare **R**oo**t**).

**Factorial**: serve para fazer um calculo fatorial, ou fatoração.

Uma observação, quando você escolhe o **import math** você terá que especificar qual ferramenta estará utilizando ou seja você terá que fazer assim:

```
1   import math
2   x = int(input('coloque um número : '))
3   y = math.sqrt(x)
4   print(f'a raiz quadrada de {x} é: {y}')
```

o comando é o math.(a ferramenta que você quer utilizar), logo você irá fazer isso com todas as outras ferramentas, assim dando a instrução correta para o computador saber o que realmente precisa ser feito.
Agora quando você especifica, nesse caso **from math import sqrt**
não será necessário fazer dar o comando **math.**
Olha como fica quando é especificado:

```
1   from math import sqrt
2   x = int(input('coloque um número : '))
3   y = sqrt(x)
4   print(f'a raiz quadrada de {x} é: {y}')
```

E você pode especificar também por exemplo se você quer utilizar mais de uma ferramenta, por exemplo:

```
1  from math import sqrt, ceil, floor
2  x = int(input('coloque um número : '))
3  y = sqrt(x)
4  print('a raiz quadrada de {} é: {}'.format(x, ceil(y)))
```

```
1  from math import sqrt, ceil, floor
2  x = int(input('coloque um número : '))
3  y = sqrt(x)
4  print('a raiz quadrada de {} é: {}'.format(x, floor(y)))
```

e claro sempre pesquise o que dá pra fazer com cada módulo do python, sempre você achará algo a mais.

E claro no site python.org você pode encontrar vários módulos, no youtube também, no google em si principalmente, e dependendo do que você quer fazer você nem precisa importar um módulo, ou seja o python talvez já tenha algum módulo que já faz isso de maneira diferente.

==========

## Fatiamento de Strings (palavras)

Já parou para pensar se existe uma possibilidade em cortar palavras em um computador assim como você fazia talvez na escola ou saber quantas letras há em uma palavra sem ter que precisar contar?
Isso tudo é possível no python graças a pequenas funções que descreverei abaixo, junto com algumas situações.

Vamos supor que você queira escrever uma pequena frase como “hello world”, mas você quer saber quantos caractéres tem essa palavra, quantos o essa palavra tem e você quer que somente “hello” fique com o inicio (H) maiusculo ficando assim Hello world.
A pergunta é, como?, como eu faço essa extração dessas informações e como irei fazer tal alteração com a frase que já foi escrita?

Aqui irão algumas funções do python para isso:

**Len()**: Serve para mostrar o tamanho ou **Len**gth (comprimento) de uma frase então essa função mostrará a partir do 0 até o tamanho maximo das letras que nesse caso é 11:
input: output: `11`

```
frase = 'hello world'
x = len(frase)
print(x)
```

Talvez você já contou e pensou: calma, nessa frase só temos 10 letras porque então deu 11?, isso se deve por que o computador leu até mesmo o espaço já que isso também ocupa memoria dentro do sistema além das letras.

**count()**: serve para contar nesse caso, letras, ou seja, em “Hello world”, podemos saber quantos O temos nessa frase dessa forma:

input:

```
frase = 'hello world'
x = frase.count('o')
print(x)
```

output:

```
2
```

Eu sei que isso parece ser meio “pra que” mas quando se pega uma frase maior dá pra fazer um pequeno script que conta a quantidade de letras por cada texto, exemplo: “esse texto tem: 5 letras “a”, 10 letras “b”, 3 letras “c”, 2 letras “d” e assim por diante...

**find()**: serve para encontrar determinadas palavras, letras como por exemplo h:

input:

```
frase = 'hello world'
x = frase.find('h')
print(x)
```

output:

```
0
```

Mas porque deu zero?, isso se deve que a contagem sempre começa do zero, ou seja o H está localizado no número zero, assim:

0 = h
1 = e
2 = l
3 = l
4 = o
5 =
6 = w
7 = o
8 = r
9 = l
10 = d

e então se contarmos desde o zero temos onze caracteres incluindo o espaço que está sendo ocupado pelo 5.

**replace()**: serve para substituir uma palavra, letra ou algo nesse tipo

input:

```
frase = 'hello world'
x = frase.replace('hello', 'goodbye')
print(x)
```

output:

```
goodbye world
```

Nesse caso foi substituido a palavra “hello”, para “goodbye”.

**upper()**: transforma tudo em maiusculo nesse caso. Exemplo (HELLO WORLD).

**lower()**: transforma tudo em minusculo. Exemplo: (hello world).

**captalize()**: transforma o inicio em maiusculo. Exemplo: (Hello world).

**title()**: transforma o inicio após espaço em maiusculo. Exemplo (Hello World).

**strip()**: remove espaços que não são uteis na string. Exemplo

input:
___Hello_World____ ( ___ = espaço, somente nesse caso.)

output:
Hello_World (foi removido o do começo e do fim).

Temos também o **rstrip()** e o **lstrip()**, **rstrip** remove os espaços da direita já o **lstrip** remove o da esquerda.

**split()**: serve para separar em “listas” as palavras, ou seja, ficaram separadas uma da outra.
Exemplo:

```
                                                    _____________________________
hello world      »      hello | world      «      | hello vai ter no total de 5  |
      ^                       ^                   | números contando do zero     |
      ^                       ^                   | e world vai ficar com 5      |
      ^                       ^                   | também                       |
      ^                       ^                    -------------------------------
      ^                       ^
Está junta       e      Vai ficar separada após usar o split().
```

Nessa pequena representação a função split vai separar e vai fazer com que cada um tenha uma quantidade própria.

**join()**: serve para adicionar alguma coisa entre cada palavra, letra como por exemplo “-” ou “+”.
exemplo:

input:

```
frase = 'hello world'
x = '-'.join(frase)
print(x)
```

output:

```
h-e-l-l-o- -w-o-r-l-d
```

Então você pode fazer algumas coisinhas com essa pequena função.

==========