

Python do básico

Feito por:
Breno Cardoso
GitHub:
[@BrenoCardoso2002](#)

Sumário

Python Básico	3
O que é Python	3
Como instalar o Python	3
Como programar em Python	4
Como executar um código Python	4
Observação pré desenvolvimento Python	5
Primeiro código em Python	5
Segundo código em Python	6
Terceiro código em Python	7
Quarto código em Python	8
Quinto código em Python	9
Código final em Python	10
Repositório com os arquivos	10

Python do básico

Aqui está um PDF contendo um tutorial introdutório de *Python*, projetado para ajudar a dar os primeiros passos no aprendizado dessa linguagem de programação.

O que é *Python*:

Python é uma linguagem de programação de alto nível, versátil e de fácil compreensão, com uma comunidade ativa e acolhedora, bibliotecas abrangentes e suporte multiplataforma, o que a torna uma escolha popular para desenvolvedores em diversas áreas.

Como instalar o *Python*:

A seguir, estão listados os passos para a instalação do *Python* em sua máquina:

1. Acesse o site oficial do *Python* em python.org.
2. Faça o *download* da versão mais recente do *Python* compatível com o seu sistema operacional.
3. Execute o arquivo de instalação que foi baixado.
4. Certifique-se de marcar a opção "*Add to Path*" para facilitar o uso do *Python* no terminal.
5. Siga as instruções fornecidas para concluir a instalação.
6. Para verificar se a instalação foi bem-sucedida, siga os seguintes passos dependendo da versão que foi instalada:
 - Digite "*python --version*" ou "*python3 --version*" no terminal.
 - Se a instalação foi concluída com sucesso, será exibido a versão instalada do *Python*.
 - Se houver algum erro ou mensagem informando que o comando não foi encontrado, a instalação não foi bem-sucedida.

Como programar em *Python*:

Existem várias opções disponíveis para programar em *Python*, e a escolha pode depender das preferências e gostos individuais. Aqui estão alguns exemplos:

1. ***IDLE***:

O *IDLE* é uma *IDE* (*Integrated Development Environment*) que já vem incluída na instalação padrão do *Python*. Ele fornece um editor simples e uma janela de console para executar seu código.

2. ***PyCharm***:

O *PyCharm* é uma *IDE* amplamente utilizada pela comunidade *Python*. Ele oferece recursos avançados, como sugestões de código, depuração, controle de versões e muito mais.

3. ***Visual Studio Code* (*VS Code*)**:

O *VS Code* é um editor de código-fonte altamente personalizável que suporta várias linguagens de programação, incluindo *Python*. Embora não seja uma *IDE* completa, pode ser estendido com uma ampla variedade de extensões para atender às suas necessidades.

É importante ressaltar que essas são apenas algumas das opções populares disponíveis, e há uma ampla variedade de *IDEs* e editores disponíveis para programar em *Python*. É recomendável explorar diferentes opções e escolher aquela que melhor se adapte às suas preferências e estilo de desenvolvimento. Cada *IDE* ou editor possui suas próprias características e recursos, então vale a pena experimentar diferentes ferramentas para encontrar aquela que atenda às suas necessidades específicas. O importante é encontrar uma ferramenta que facilite sua produtividade e ofereça suporte às funcionalidades desejadas durante o desenvolvimento em *Python*.

Como executar um código *Python*:

Para executar um arquivo *Python*, é necessário salvá-lo com a extensão `".py"`.

No *PyCharm* ou *VS Code*, geralmente há um botão de reprodução que permite a execução direta do código.

No caso da execução pelo terminal, você deve digitar o comando `"python"` ou `"python3"`, dependendo da versão instalada, seguido pelo nome do *arquivo.py*. É importante garantir que o terminal esteja na pasta onde o arquivo a ser executado está localizado.

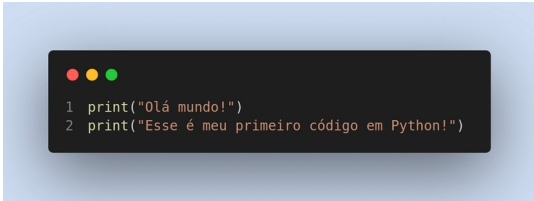
Observação pré desenvolvimento *Python*:

Antes de começar a escrever códigos em *Python*, é importante observar algumas informações essenciais:

1. Sensibilidade a maiúsculas e minúsculas:
Python é uma linguagem "*case-sensitive*", ou seja, faz distinção entre caracteres maiúsculos e minúsculos. É importante ter cuidado ao utilizar letras maiúsculas e minúsculas em nomes de variáveis, funções e comandos, pois erros podem ocorrer se a capitalização não estiver correta.
2. Indentação obrigatória:
Python é uma linguagem que exige a utilização de indentação (reco de linhas) para delimitar blocos de código. A indentação é fundamental para definir a estrutura e organização do código. Certifique-se de manter uma indentação consistente ao escrever seu código, usando espaços ou tabulações de forma consistente em cada bloco. A falta de indentação correta pode resultar em erros de sintaxe e comportamento inesperado do código.

Fique atento a esses pontos mencionados acima, pois eles desempenham um papel fundamental na correta execução do seu código *Python* e na prevenção de erros.

Primeiro código em *Python*:



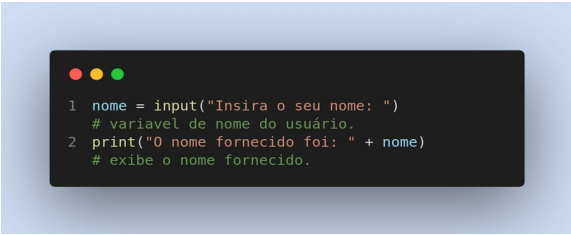
No código acima, é mostrada uma imagem com dois exemplos de uso da função ***print()*** em *Python*.

A função ***print()*** é uma função nativa que é utilizada para exibir mensagens na tela.

Vale salientar que cada mensagem será exibida em uma linha, pois elas estão sendo passadas como argumentos para chamadas diferentes da função ***print()***.

Esse é um exemplo simples que demonstra como a função ***print()*** pode ser usada para exibir dados na tela durante a execução de um programa *Python*.

Segundo código em *Python*:



```
1 nome = input("Insira o seu nome: ")  
  # variavel de nome do usuário.  
2 print("O nome fornecido foi: " + nome)  
  # exibe o nome fornecido.
```

O código acima exibe um exemplo prático com explicações sobre os conceitos a seguir:

1. Criação de variáveis:

Na criação de uma variável em *Python*, primeiro você define o nome da variável e, em seguida, atribui um valor a ela.

No exemplo apresentado, a variável "nome" é criada e recebe o valor fornecido pelo usuário por meio da função *input()*.

2. Concatenação de variáveis:

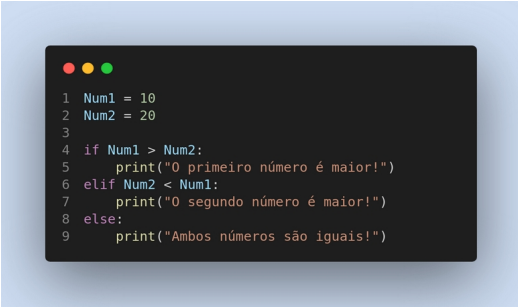
Em linguagens de programação, a concatenação refere-se à junção de um texto com o valor de uma variável. No exemplo, o valor da variável "nome" é concatenado com um texto pré-definido, resultando na exibição do nome fornecido pelo usuário juntamente com a mensagem "O nome fornecido foi:".

3. Comentários:

Em *Python*, os comentários são feitos utilizando o caractere "#" no início da linha. Tudo que estiver após o "#" não será considerado como código executável. Comentários são úteis para adicionar explicações e fazer anotações no código, tornando-o mais legível e compreensível.

Esses tópicos fornecem uma breve explicação sobre as principais características abordadas no código. Entender esses conceitos é fundamental para escrever e compreender programas em *Python*.

Terceiro código em *Python*:



```
1 Num1 = 10
2 Num2 = 20
3
4 if Num1 > Num2:
5     print("O primeiro número é maior!")
6 elif Num2 < Num1:
7     print("O segundo número é maior!")
8 else:
9     print("Ambos números são iguais!")
```

O código acima apresenta uma condição de comparação em *Python*.

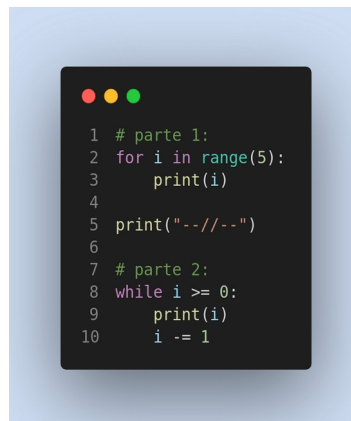
Em *Python*, usamos o comando "**if**" para realizar uma comparação específica. No exemplo, comparamos os valores das variáveis "Num1" e "Num2". Se a condição especificada no "**if**" for verdadeira (ou seja, se Num1 for maior que Num2), a mensagem "O primeiro número é maior!" será exibida.

Caso a condição do "**if**" não seja atendida, usamos o comando "**elif**" para realizar outra comparação específica, que é diferente da condição do "**if**". No exemplo, se a condição do "**elif**" for verdadeira (ou seja, se Num2 for menor que Num1), a mensagem "O segundo número é maior!" será exibida.

Por fim, se nenhuma das condições anteriores for verdadeira, usamos o comando "**else**" para executar um bloco de código para qualquer caso contrário. No exemplo, se nenhum dos números for maior que o outro, a mensagem "Ambos números são iguais!" será exibida.

Esses comandos (**if**, **elif** e **else**) permitem realizar diferentes ações com base em condições específicas, tornando o código mais flexível e adaptável a diferentes situações.

Quarto código em *Python*:



```
1 # parte 1:
2 for i in range(5):
3     print(i)
4
5 print("---//---")
6
7 # parte 2:
8 while i >= 0:
9     print(i)
10    i -= 1
```

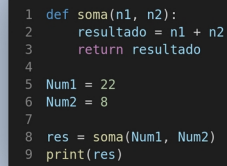
O código acima apresenta duas maneiras de criar um laço de repetição em *Python*.

A primeira maneira é utilizando o comando "**for**". No exemplo, é criada uma variável chamada "*i*" que percorre a sequência gerada pela função "*range(5)*". A função "*range()*" gera uma sequência de números em um determinado intervalo. Nesse caso, o *loop* é executado cinco vezes, e em cada iteração, o valor de "*i*" é impresso.

A segunda maneira de criar um laço de repetição é utilizando o comando "**while**". No exemplo, a variável "*i*" é inicializada previamente e o bloco de código dentro do "**while**" é executado repetidamente enquanto a condição "*i* >= 0" for verdadeira. A cada iteração, o valor de "*i*" é impresso e, em seguida, é decrementado por 1.

Essas duas estruturas de repetição permitem executar um bloco de código várias vezes, mas com lógicas diferentes. O "**for**" é usado quando se conhece a quantidade exata de iterações, enquanto o "**while**" é mais flexível e permite executar o *loop* até que uma condição seja satisfeita.

Quinto código em *Python*:



```
1 def soma(n1, n2):  
2     resultado = n1 + n2  
3     return resultado  
4  
5 Num1 = 22  
6 Num2 = 8  
7  
8 res = soma(Num1, Num2)  
9 print(res)
```

O código acima demonstra o uso de uma função em *Python*.

Em *Python*, as funções são blocos de código que podem ser reutilizados para realizar uma tarefa específica. As funções podem receber argumentos de entrada e retornar um valor ou executar uma ação.

No exemplo, a função chamada "soma" é definida. Ela recebe dois argumentos, "n1" e "n2", e realiza a soma desses dois valores, armazenando o resultado na variável "resultado". Em seguida, o resultado é retornado pela função.

Após a definição da função, são criadas duas variáveis, "Num1" e "Num2", que armazenam os valores a serem somados. Em seguida, a função "soma" é chamada, passando essas duas variáveis como argumentos. O valor retornado pela função é armazenado na variável "res".

Por fim, o valor armazenado em "res" é impresso na tela usando o comando *print()*.

Essa abordagem de dividir o código em funções permite modularizar e reutilizar partes do código, tornando-o mais organizado e fácil de manter.

Código final em Python:

```
1 # função que verifica se o número é par ou ímpar:
2 def verifica_par(numero):
3     if numero % 2 == 0:
4         return True
5     else:
6         return False
7
8 indice = 1 # variável do índice do laço de repetição:
9
10 # laço de repetição:
11 while indice <= 5:
12     num = input("Insira o " + str(indice) + "º número: ")
13     numero = int(num)
14     if verifica_par(numero):
15         print(str(num) + " é um número par!")
16     else:
17         print(str(num) + " é um número ímpar!")
18     indice += 1
19     print() # print para ter uma linha em branco entre cada repetição.
```

O código acima é um exemplo que utiliza diversos conceitos e métodos discutidos anteriormente.

Primeiramente, é definida uma função chamada "verifica_par" que recebe um número como argumento e verifica se ele é par ou ímpar. A função retorna *True* se o número for par e *False* caso contrário.

Em seguida, é declarada uma variável chamada "índice" que serve como o índice do laço de repetição.

O código entra em um laço de repetição utilizando o comando "*while*". O laço é executado enquanto o valor de "índice" for menor ou igual a 5. Em cada iteração do laço, o usuário é solicitado a inserir um número, que é armazenado na variável "num". Esse valor é convertido para um número inteiro utilizando a função *int()* e atribuído à variável "numero".

A função "verifica_par" é chamada, passando o valor de "numero" como argumento. Dependendo do resultado da verificação, é exibida uma mensagem indicando se o número é par ou ímpar.

Após isso, o valor de "índice" é incrementado em 1 e é utilizado o comando *print()* para adicionar uma linha em branco entre cada repetição do laço.

Esse código exemplifica a utilização de diferentes conceitos, como definição de função, uso de condicionais, laço de repetição e conversão de tipos de dados. A combinação desses elementos permite criar programas mais complexos e flexíveis em *Python*.

Repositório com os arquivos:

Os códigos presentes nesse documento PDF estão disponíveis no *GitHub*, no perfil [@BrenoCardoso2002](https://github.com/BrenoCardoso2002).

O link do repositório é o seguinte:

https://github.com/BrenoCardoso2002/Python_do_basico.