Sumário

Capítulo 1

Primeiros passos com python!

1.1 Por que python?

Python é uma linguagem simples, poderosa e extremamente versátil. Ela pode ser usada em diversas áreas: ciência de dados, desenvolvimento de jogos, criação de sites, aplicativos, aprendizado de máquina, automações, administração de sistemas e muito mais. Se o foco do seu projeto não for performance extrema, como é o caso de linguagens como C, é bem provável que você consiga resolver com Python.

Em outras palavras, se você está começando sua jornada no mundo da programação, Python é como um bote salva-vidas que pode te ajudar a atravessar qualquer mar de códigos! (risos)

1.2 O que é programação?

Programação é o ato de escrever instruções em forma de texto (código) que o computador deve seguir. Essas instruções precisam ser simples, passo a passo, e, o mais importante, sem ambiguidades!

Você já ouviu aquela piada do programador? A esposa diz: "Amor, quero que você vá ao mercado e traga 2 ovos. Se tiver leite, traga 6."O homem vai até o mercado, encontra leite, mas volta com 6 ovos e nenhum leite. Pode parecer engraçado, mas eu te pergunto: a instrução estava clara?

1.2.1 Uma breve história da ciência da computação: como chegamos até aqui

Embora o autor deste livro não seja exatamente um entusiasta de histórias antigas, é impossível ignorar como a ciência da computação percorreu um longo caminho para chegar ao que temos hoje.

Nos primórdios, programar um computador era uma tarefa bem diferente do que é atualmente. Os programas eram escritos em cartões perfurados, que precisavam ser inseridos em grandes máquinas para processamento. Cada pequeno erro, como trocar a ordem de um cartão, poderia comprometer todo o programa. Além disso, os computadores eram imensos, ocupando salas inteiras, e a ideia de ter um em casa era impensável.

Com o tempo, evoluímos para os chamados terminais burros, dispositivos conectados a servidores centrais. Embora esses terminais permitissem aos usuários escrever comandos, o trabalho pesado era feito por computadores maiores e mais poderosos. Ainda assim, a experiência deixava a desejar: havia um atraso considerável entre o pressionar de uma tecla e a resposta do servidor.

Hoje, a realidade é completamente diferente. Graças à evolução da computação, você pode escrever centenas ou milhares de linhas de código e obter resultados em segundos ou minutos. O que antes exigia salas inteiras e equipamentos especializados agora pode ser feito em um laptop ou até mesmo em um smartphone.

E é impossível falar da história da computação sem mencionar Alan Turing: matemático brilhante, pioneiro da ciência da computação, e uma figura histórica cujas ideias revolucionaram o campo. Seu trabalho nos ajuda a entender como chegamos a um ponto em que programar se tornou acessível a todos. Obrigado, Turing!

1.3 Ferramentas e como instalá-las

Bom, a partir daqui eu assumo que você tenha acesso à algum computador (pode ser até um chromebook ou coisa que o valha), de todo modo, essas serão nossas principais ferramentas:

- Visual Studio Code
- Python
- Pycharm

1.3.1 Instalando o VSCode

A instalação do VSCode é bem simples. Ao baixar o arquivo, basta seguir os passos do instalador. Durante o processo, você verá algumas opções, como a de adicionar o VSCode ao contexto da pasta atual (uma opção muito útil, pois facilita a abertura de projetos diretamente do explorador de arquivos). Não é obrigatório, mas pode ser interessante para facilitar sua vida mais tarde.

1.3.2 Instalando o Python

Baixe o Python; quando for instalar, se você tiver permissões de administrador, você pode optar por instalar o Python para todos os usuários do computador.

Importante

Não se esqueça de marcar a opção "Add Python to PATH"(Adicionar Python ao PATH) durante a instalação. Isso é essencial para que você possa rodar o Python diretamente do terminal (sem precisar navegar até a pasta onde o Python está instalado).

1.3.3 Instalando o PyCharm

Se você optar por usar o PyCharm, cuidado! Ao entrar no site da jetbrains para fazer o download, você verá duas opções: PyCharm Professional (pago) e PyCharm Community Edition (GRATUITO). Role a página até encontrar a versão Community Edition e baixe-a. Essa versão é perfeita para quem está começando.

1.4 Meu primeiro programa

Agora que já sabemos o que é programação, estamos prontos para escrever nossa primeira linha de código! Digite o seguinte comando no seu editor de texto:

```
1 print("Ola Mundo!")
```

Código 1.1: Olá mundo em python

Salve o arquivo com o nome "ola_mundo.py"(com a extensão .py, que é fundamental para que o Python reconheça o arquivo como um script Python). Agora, você pode rodar o seu código de duas maneiras:

1. No editor de texto: Normalmente, você encontrará um botão de "play"ou "executar"para rodar o script diretamente no editor.
2. No terminal: Se preferir usar o terminal, abra a pasta onde o arquivo está salvo e digite o seguinte comando: python ola_mundo.py

Importante

A extensão .py é essencial para que o arquivo seja identificado como um script Python.
Além disso, a convenção do Python para nomes de arquivos é utilizar o snakecase — ou seja, todas as palavras em minúsculas, separadas por underscores (sublinhados).
Então, lembre-se: ao nomear seus arquivos, use essa convenção para manter o código organizado e legível.

O que faz o nosso primeiro *"Olá, Mundo"*?

O comando que escrevemos, `print("Olá, Mundo!")`, tem uma função bem simples: ele imprime (mostra) a mensagem "Olá, Mundo!"na tela do terminal.

É isso! Parece algo pequeno, mas é o primeiro passo para aprender a conversar com o computador.

Apesar de ser uma linha de código tão simples, ela já nos ensina um dos fundamentos mais importantes da programação: a saída de dados — ou seja, como fazer o computador apresentar informações para nós.

Mas e os programas mais complexos?

Acredite: todos os programas, por mais avançados ou sofisticados que sejam, são compostos pelas mesmas estruturas básicas. Programar é como construir com blocos de montar: aprendemos a usar essas peças fundamentais, e, aos poucos, conseguimos criar coisas incríveis!

1.5 Estruturas que compõem um programa

Aqui estão as estruturas que formam a base de qualquer programa:

Componentes de um programa

Variáveis: Para armazenar informações.
Entrada de dados: Para capturar dados do usuário ou de outras fontes.
Saída de dados: Para exibir informações (como o nosso "Olá, Mundo!").
Processamento: Realizar cálculos e transformar dados.
Estruturas condicionais (if): Para tomar decisões com base em condições.
Laços de repetição (for, while, do-while): Para executar uma ação várias vezes.
Funções: Blocos de código reutilizáveis que realizam tarefas específicas.
Classes: Para organizar e estruturar programas maiores (usando conceitos de orientação a objetos).
Tratamento de exceções: Para lidar com erros de forma controlada.
Leitura e escrita de arquivos: Para salvar ou acessar informações armazenadas no computador.

Essas peças são como os "ingredientes básicos"da programação. Com elas, você poderá criar desde simples scripts até sistemas complexos. Utilizando o Python como uma Calculadora

1.6 Python como calculadora

Antes de introduzirmos o conceito de variáveis, vamos explorar um dos usos mais simples e práticos do Python: como uma calculadora! Isso permite que você execute operações matemáticas diretamente e veja os resultados na hora.

1.6.1 Passo 1: Abrindo o Python no Modo Interativo

No Windows, pressione a tecla Windows e digite "python"na barra de busca. Abra o programa Python 3.X (a versão que você instalou). Isso abrirá o modo interativo, onde você pode digitar comandos e ver os resultados instantaneamente.

1.6.2 Passo 2: Testando Operações Matemáticas

Digite os comandos abaixo e observe os resultados. Esses exemplos mostram diferentes operações que você pode fazer no Python:

```
3 + 3                # soma
4 * 3                # multiplicacaoo
12 / 4               # divisao (resultado com casas
   decimais)
12 // 4              # divisao inteira
3 ** 2               # potencia (3 elevado a 2)
26 - 5               # subtracao
3 * 4 - 2 + 13 / 3 # combinacao de operacoes: nesse caso
   a ordem segue o acronimo PEMDAS (mais sobre isso a
   seguir.)
3 * (4 - 2)          # uso de parenteses para alterar a
   ordem das operacoes.
```

Código 1.2: Python como calculadora.

1.6.3 O que aprendemos com isso?

O Python segue a ordem de precedência matemática, como em uma calculadora comum. Multiplicações e divisões são realizadas antes de somas e subtrações, a menos que você use parênteses para forçar uma ordem diferente.
Você pode experimentar diferentes números e combinações de operações para entender como o Python processa cada comando.

Dica: Se você vir algo como `>>>` no terminal, é o Python esperando pelo próximo comando. Aproveite e teste à vontade!

1.7 Variáveis

O que são? Onde vivem? Para que servem? Tudo isso será respondido.

> *"Tudo será revelado"* — Mortred, Warcraft 3

O que são variáveis?

Você pode pensar em uma variável como um recipiente — como um balde, pote ou caixa — que serve para armazenar algo. Cada variável possui:

- Um **tipo**: Define o que a variável pode armazenar (números, texto, valores booleanos, etc.).
- Um **nome**: A identificação da variável, que você escolhe.
- Um **valor**: O conteúdo que ela armazena.

Por exemplo:

```
num = 3
```

Código 1.3: código simples

Aqui:

- O **tipo** é um inteiro (número sem casas decimais).
- O **nome** da variável é num.
- O **valor** armazenado é 3.

Isso permite realizar operações e até alterar o valor armazenado na variável. Por exemplo, poderíamos somar algo a num ou redefinir seu valor.

Onde *"vivem"* as variáveis?

As variáveis "vivem"na memória do computador, mas não no HD ou SSD! Esses são exemplos de memórias secundárias, usadas para armazenamento permanente. As variáveis de programas em execução vivem na memória RAM (Random Access Memory ou Memória de Acesso Aleatório) – Essa é a memória volátil usada enquanto o programa está rodando.

1.7.1 Nomes de variáveis: o poder está em suas mãos!

Uma das coisas legais sobre variáveis é que você pode dar o nome que quiser a elas, desde que siga algumas regras básicas do Python, basicamente: começar com uma letra ou sublinhado, sem espaços.

Por exemplo:

```
rock_n_roll_eh_bom = True
print(type(rock_n_roll_eh_bom))
# devera mostrar <class 'bool'>
# ou seja, um tipo booleano, mais sobre classes adiante!
```

Código 1.4: exemplo de variável

Aqui, criamos uma variável com um nome bem expressivo!

- O **nome** da variável é rock_n_roll_eh_bom
- O **tipo** é booleano (True ou False).
- O **valor** é True (sim, rock'n'roll é bom!).

Isso mostra como nomes de variáveis podem tornar seu código mais legível e divertido!

De fato, você pode até verificar o tipo da variável usando a função type como demonstrado no código acima! não apenas veja o código, tente executá-lo na sua máquina também!

Resumo:

Variáveis são recipientes que armazenam valores. Elas têm um tipo, nome e valor. Vivem na memória RAM enquanto o programa está rodando. Você pode escolher nomes significativos para elas, tornando seu código mais claro e até criativo.

Observação

Embora tecnicamente seja possível usar acentos ou caracteres especiais em nomes de variáveis, não é recomendado! Isso pode causar problemas, especialmente ao trabalhar com bibliotecas externas ou ambientes que não reconhecem esses caracteres corretamente. O autor deste livreto recomenda fortemente que você aprenda inglês e não cometa heresias ao nomear suas variáveis! Este livro é apenas o começo de uma boa jornada!

1.7.2 Quais tipos existem no python?

A resposta curta é que existem seis tipos, por padrão na linguagem.

A resposta longa é: quantos você quiser.

Vamos aos seis tipos principais, são eles:

1. **Booleano** – aceita apenas True [Verdadeiro] ou False [Falso]
2. **Inteiro** – número inteiro
3. **Float** – ponto flutuante (vulgarmente conhecido como decimal)
4. **String** – texto, ou "cadeia de caracteres"
5. **None** – um tipo que representa "vazio"ou ausência de valor
6. **Complexo** – um tipo específico pra trabalhar com números complexos, não exploraremos neste livro.

para tornar toda essa discussão sobre tipos mais "palpável", tente executar os códigos abaixo.

```
# Inteiros
print(type(200))
print(type(0))
print(type(-4400))

# Booleanos
```

```
7  print(type(True))
8  print(type(False))
9
10 # Ponto flutuante
11 print(type(3.1415))
12 print(type(206.24))
13 print(type(-30005.08))
14
15 # None
16 print(type(None))
17
18 # String
19 print(type("testando, ola mundo"))
20 print(type("aaaaaaaatum"))
21
22 # Complexo
23 # O 'j' significa a unidade imaginaria em python!
24 print(type(13j)) #numeros complexos precisam ser '
     posfixados' com a letra j
25 print(type(3 + 4j)) # outro numero complexo
26
27
```

Código 1.5: Verificando os tipos de variáveis em python

Veja que, todos esses tipos aparecem como <class 'tipo'>

Pois não seria interessante se você pudesse criar seu próprio tipo? O que é uma "classe", afinal?
Calma, vamos explorar isso em capítulos futuros! Por enquanto, focaremos na programação imperativa: um comando após o outro, instruindo a máquina passo a passo.

1.7.3 E eu com os tipos?

O que eu quero demonstrar aqui, é que uma boa parte do seu trabalho como programador será receber dados, transformá-los e apresentar ao usuário algo de relevante com eles, considere o seguinte trecho de código:

```
1 idade_do_joao = 14
2 # quantos anos joao tera daqui a 8 anos?
3 idade_do_joao = idade_do_joao + 8
4 print("joao tera", idade_do_joao, "anos, daqui a 8 anos")
```

Código 1.6: uma conta simples

Aqui, acontece que você está somando 8 a uma variável que já é um número inteiro, ou seja, não é necessário fazer nenhum tipo de conversão – dentro do

print, a variável "idade_do_joao"é exibida, repare que, como ela está entre vírgulas, ela é **concatenada** ("juntada") com espaços ao resto da string.

Suponha, no entanto, que eu queira receber do próprio usuário qual a idade dele, e, só então, calcular quantos anos ele terá daqui a 8 anos.

```
1 idade_do_usuario = int(input("Quantos anos voce tem hoje?
    "))
2 print("daqui a 8 anos, voce tera", idade_do_usuario + 8)
```

Código 1.7: Calculando a idade futura do usuário

Salve o código acima como "idade_futura_do_usuario.py"e execute ele, se tudo correr bem, o programa deve pedir a idade do usuario, você vai digitar um número qualquer, do tipo 25, e ele vai mostrar quantos anos você terá daqui 8 anos, que no caso seria 33.

Nesse código, introduzimos 2 coisas diferentes: estamos fazendo uma conversão de tipos, um *type casting* no valor que de entrada que o usuário informou através da função input e, subsequentemente atribuindo esse valor, agora convertido pra inteiro, pra variável idade_do_usuario.

Basicamente, ao executar esse código o programa vai perguntar a idade com a seguinte mensagem: "Quantos anos você tem hoje?", depois disso, ele vai converter essa entrada de dados para um inteiro, portanto, nesse primeiro momento, o usuário não pode digitar algo como "eu tenho 15", pois o programa espera somente um número – em outras palavras, quaisquer coisas digitadas naquele momento que não sejam números, ocasionarão em uma exceção, um erro de valor (mais sobre exceções adiante!)

Observação

Isso que fizemos no Código 1.7 é denominado **composição de funções**. Repare: uma função lê a entrada do teclado, a outra converte para inteiro, as duas juntas permitem o calculo e a apresentação para o usuário.

Isso é extremamente comum no mundo da programação, você não precisa ser nenhum gênio da matemática pra compor suas funções.

Licença ©