Capítulo 1

Introdução

O que é programação?

Programar é dar uma série de instruções ao computador. Esta série de instruções é conhecida como algoritmo.

É comum, no brasil, começarem ensinando programação por pseudo-linguagens, ou seja, utilizando português para criar algoritmos. Exemplo:

```
dê dois passos para a frente,
tente abrir a porta
se conseguir, abra a porta
se não conseguir, vire para a esquerda
```

E depois ensinam através do portugol, que é uma linguagem que utiliza o português como base, para facilitar a compreensão de conceitos sem aprender palavas em inglês. Porém, eu discordo desta abordagem e prefiro utilizar uma linguagem que você usará no dia-a-dia, pois a sintaxe da maioria das linguagens é parecida. Assim você não precisa reaprender toda a sintaxe para começar a trabalhar num projeto real.

Preciso saber matemática?

Um motivo de ansiedade ao redor de quem considera aprender a programar é o *mito* de que saber matemática é uma necessidade. Sim, eu disse *mito*. E digo isto pois na maioria das vezes programação não requer nada além de matemática básica. É importante entender que dentro da programação existem diversas profissões. Algumas vão requerer matemática, outras não. Novamente: a maioria não vai. Só porque algo envolve números, não quer dizer que você precise de matemática para aprender.

Eu acho que estou muito velho pra aprender a programar

Outro *mito* bastante comum é de que só consegue aprender a programar bem, aqueles que aprenderam cedo. Programação se aprende fazendo, praticando. Você cresce com cada dia

que pratica. Ninguém nasce um programador. É uma habilidade adquirida através de estudo e prática.

Por que python?

Python é uma linguagem de *"uso geral"* , o que significa que é uma linguagem usada em diversas áreas, incluindo ciência de dados, machine learning, desenvolvimento de software, desenvolvimento web, automação, entre outras. Além de ser uma linguagem amplamente utilizada, sua sintaxe é muito parecida com o inglês, o que a torna muito menos intimidadora para quem está começando.

Sobre o curso

O curso foi desenvolvido de tal maneira que a cada módulo adicionaremos mais matéria. Sendo assim, tudo o que vermos voltará a ser utilizado, por isso é crucial que você avance de módulo apenas quando tiver entendido bem a matéria.

Nós visitaremos outros tópicos além de python no decorrer do curso, mesmo que não tão a fundo, pois precisaremos desse conhecimento para fazermos alguns de nossos projetos.

Não se limite aos exercícios do curso. Programação é como LEGO. É um exercício de criatividade. Aqui você vai aprender todas as ferramentas e conhecimentos necessários para criar o que imaginar, além da habilidade de aprender uma outra linguagem.

Sobre os exercícios

Existem diversas maneiras de se fazer a mesma coisa na programação. Alguns códigos são mais otimizados que outros, mas desde que tudo funcione, tudo bem. As respostas dos exercícios podem ser encontradas neste repositório.

Tente fazer do seu jeito e, caso não consiga, veja minhas respostas.

Bonus: Como funciona um computador?

Cada peça de um computador tem um trabalho específico. Aqui quero esclarecer um pouco mais o que cada peça de hardware faz, de maneira superficial. Isto vai nos ajudar a esclarecer umas coisas mais pra frente.

Placa Mãe

Conecta todas as peças do computador e as fazem funcionar em conjunto

CPU | Processador

É o cérebro do computador. É capaz de processar uma quantidade imensa de informações por segundo.

GPU | Placa de Video

Processa as imagens e videos do computador, como jogos, seriados, e a imagem do monitor.

HDD | SSD | NVMe

Está é a *memória de longo prazo* do computador. Os dados são **persistentes** , ou seja, mesmo depois de desligar o computador, os dados salvos no HD continuarão existindo. É onde você guarda as fotos, videos, programas, jogos, tudo que há no seu computador.

RAM | Random Access Memory

A memória RAM é a *memória de curto prazo* do computador. Ela é extremamente rápida comparada com o HD, porém, temos uma quantidade menor disponível. Toda vez que rodamos um programa, abrimos um video, fazemos algo, os dados necessários para este programa rodar ficam armazenados na memória RAM, quando fechamos estes programas, eles são removidos da memória. Por conta disso, quando um novo programa é aberto, seus dados serão colocados onde houver espaço disponível. A localização dos dados não é permanente.

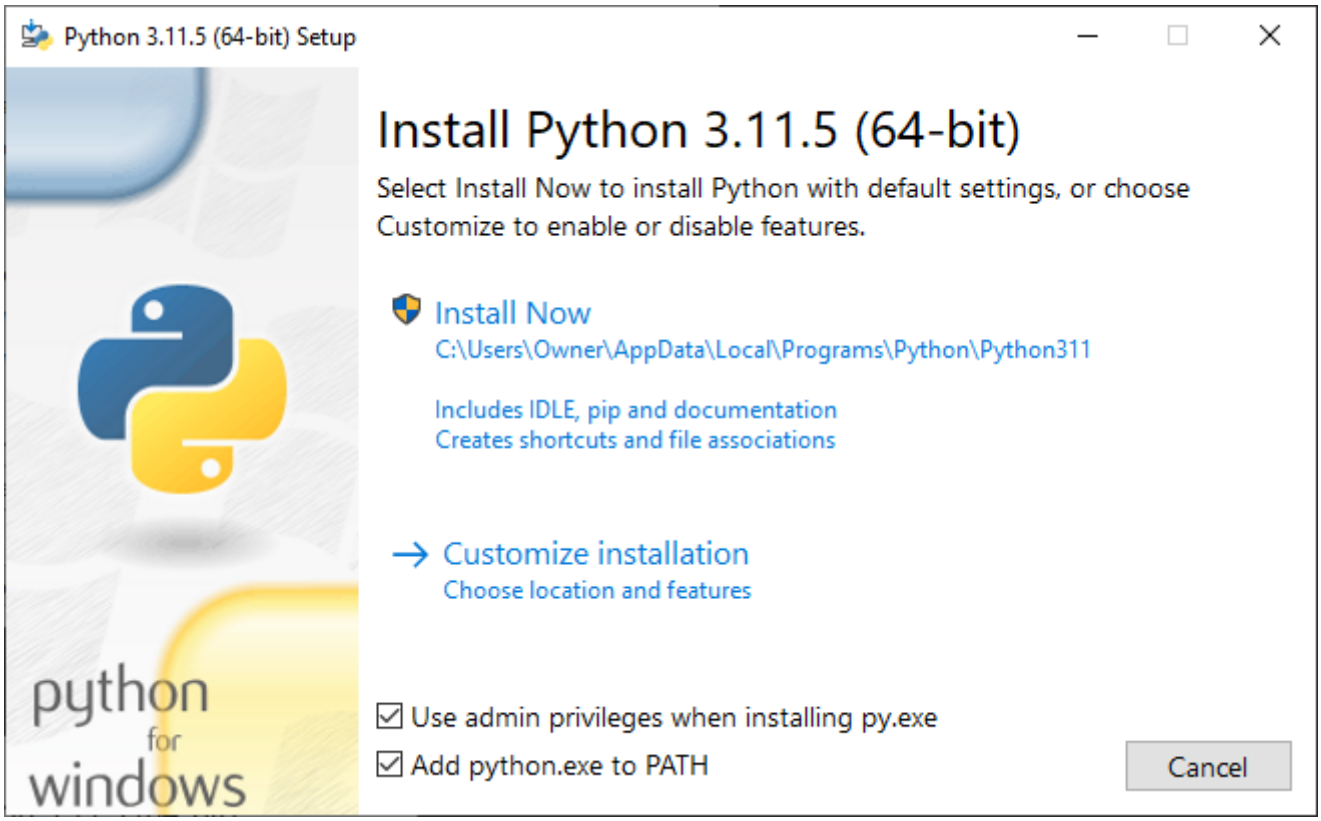
Download e instalação de python

O download de python pode ser feito através do seu site oficial.

> ***Importante:*** Faça o download de python 3 (ou a versão mais atual) para garantir que os códigos deste curso funcionem corretamente.

Windows

A instalação no windows é simples, como de qualquer outro programa. Selecione "Add python.exe to PATH" e depois é só clicar em "próximo" e esperar.



> ***Importante:*** Marcar "Add python.exe to PATH" antes de instalar.

MacOS

Para a instalação no MacOS, recomendo este link do freeCodeCamp. Fique à vontade para procurar no google ou no youtube por outros guias mais claros.

Linux

Python já vem instalado por padrão na maioria das distribuições Linux. Recomendo não atualizar pois algumas bibliotecas podem quebrar devido à diferenças entre as versões.

Instalação da IDE

Uma vez instalado, python pode ser rodado através do terminal do computador. Porém. para esta primeira parte do curso, nós instalaremos uma IDE (*Integrated Development Environment*) para programar.

> ***Nota:*** Você pode escrever seu código até mesmo no Microsoft Word ou Bloco de Notas, e executá-lo através do terminal. Um IDE é um software feito especificamente para rodar determinada linguagem, já vindo preparado e otimizado para isto. Existem outros programas, editores de texto e outras IDEs que você pode utilizar.

PyCharm Community Edition

Eu utilizarei o PyCharm como meu IDE para esta primeira parte do curso. O download pode ser feito através deste link.

> ***Nota:*** Existem duas versões, PyCharm Professional Edition, que é paga e PyCharm Community Edition, que é gratuita. Nós utilizaremos a Community Edition.

Uma vez instalado, crie um novo projeto e espere um bocadinho. A primeira vez que o projeto é aberto a IDE precisa de um tempinho para organizar as coisas.

Visual Studio Code

VSCode é um editor de texto otimizado para programação e utiliza plugins para melhorar a vida do programador. Caso esteja utilizando um computador mais antigo ou lento, recomendo utilizar o Visual Studio Code com o plugin de python. É fácil de encontrar no google como instalar plugins. O download pode ser feito neste link.

Uma vez instalado, crie um arquivo `nome_do_arquivo.py`.

Posteriormente nós utilizaremos o VSCode para alguns projetos.

Capítulo 1: O básico

Meu primeiro programa

Por padrão, a primeira coisa que fazemos para confirmarmos que um programa funciona, é dizer *"Hello, world"* , e é isso que vamos fazer. Alguns IDE's ou editores de texto já criam um arquivo python por padrão ao iniciar um projeto, outros não. Caso o arquivo não tenha sido criado, crie um arquivo chamado "hello.py". O `.py` é o que define que isto é um arquivo python. Escreva:

```
print("Hello, world")
```

> ***Nota:*** É importante utilizar as aspas na mensagem.

E execute seu programa clicando no botão ***play*** no topo da tela:

Este deverá ser o resultado:

```
hello, world

Process finished with exit code 0
```

Pronto! Você criou seu primeiro programa em python. Simples, não é?

```
print()
```

É uma função que imprime na tela os dados dentro dos parênteses.

> ***Nota:*** Nós estudaremos funções a fundo posteriormente. Por enquanto, não ligue pra elas.

Linguagens compiladas x interpretadas

Você pode ter ouvido anteriormente que computadores não entendem nada além de zeros e uns. E isto é verdade, eles só entendem código binário. Então como é possível que o computador tenha entendido o que eu escrevi? Por trás dos panos, python lê o código e "explica" para o computador o que fazer.

Existem dois tipo de linguagens, as **compiladas** e as **interpretadas**.

Linguagens como java, C e C++ são compiladas, ou seja, o código escrito é convertido para uma linguagem que o computador entende e depois nós podemos executar o programa. Nós compilamos uma vez e executamos muitas vezes.

Outras linguagens, como python e javascript, são interpretadas. Toda vez que rodamos o código, o interpretador o lê e "explica" para o computador o que fazer, de maneira que ele entenda. Toda vez que rodarmos o código, ele será lido e explicado novamente. Quando abrimos um site, por exemplo, o browser interpreta o código do site e o monta na tela.

Variáveis | Variables

Uma variável é um espacinho na memória (RAM) onde você guardará um dado. Este dado pode ser um nome, uma data, um número, etc. Soa confuso, eu sei. Vamos deixar um pouco mais claro com um exemplo. Criaremos uma variável chamada `message`:

```
message = "Hello, world"
print(message)
```

Neste exemplo, estamos separando um espacinho na memória e chamando de `message`, depois pegamos o dado `Hello, world` e colocamos dentro dela. Em seguida, chamamos a função `print()` novamente:

```
hello, world

Process finished with exit code 0
```

Como podemos ver, temos o mesmo resultado. Python imprimiu o valor atribuído à variável `message`.

> ***Importante:*** O sinal de igual (=) não é o igual da matemática. O igual comparativo/matemático nós veremos mais pra frente. Em python (=) é chamado de "Assign operator" ou operador de atribuição. Ele pega o valor do dado e atribui a uma variável.

Tá, e onde isto é usado? Pensa nos jogos que você já jogou. O número de vidas, seu HP, os nomes dos personagens, seus atributos, sua munição. Tudo isso é guardado dentro de variáveis.

Variáveis podem ter qualquer nome. Porém, existem algumas regras a serem seguidas ao escolher seu nome:

- O nome das variáveis não pode conter espaços.
- Podem conter apenas letras, números e underlines, mas podem apenas começar com letra ou underline. Não podem começar com números.
- O ideal é que seus nomes sejam descritivos e ajudem a entender o tipo de informação que ela guarda.
- Evite usar palavras reservadas por python, como o print que vimos anteriormente.

> ***Nota:*** Existem alguns estilos de nomenclatura. Entre eles estão:
>
> - **camelCase:** A primeira palavra é com letra minuscula e todas as outras são com letra maiúscula
> - **PascalCase:** A primeira letra de cada palavra começa com letra maiúscula
> - **snake_case:** Todas as palavras são com letra minúscula e separadas por underline
> - **kebab-case:** Todas as palavras são com letra minúscula e separadas por hífen
>
> Python recomenda *snake_case*.

Python lê o código de cima para baixo

As linguagens de programação, em geral, leem código de cima para baixo. No código a seguir, tentamos imprimir uma variável que ainda não foi lida por python:

```
print(f"Hello, {nome}")
nome = "João"
```

Quando tentamos rodar o código, python levanta um erro:

```
Traceback (most recent call last):
  File "/home/joao/PycharmProjects/test_project/main.py", line 2, in <module>
    print(f"Hello, {nome}")
NameError: name 'nome' is not defined. Did you mean: 'None'?

Process finished with exit code 1
```

Aqui python nos dá algumas informações sobre onde acredita estar o erro. Ele nos informa o arquivo e a linha. Depois nos diz o erro que encontrou:

```
NameError: name 'nome' is not defined. Did you mean: 'None'?
```

Python nos diz que a palavra "nome" não foi definida. Ele não a conhece, não sabe o que quer dizer. E ainda pergunta "Você quis dizer: 'None'"?

A importância dos erros

Quando python não entender o que você está tentando fazer, ele vai levantar um erro e te comunicar o que há de errado, ou tentar. Leia o erro com atenção e não tenha medo de jogá-lo no google. Tão importante quanto aprender a programar, é aprender a "desbugar", ou "fazer debug", que é o ato de resolver problemas no seu código.

Ninguém sabe tudo sobre uma linguagem. Quanto mais a utilizamos, mais aprendemos. Mas mesmo assim, é um tópico muito amplo. Por isso praticamente tudo na programação é documentado. Na documentação encontramos, em detalhes, como utilizar certas ferramentas e como a linguagem roda por trás dos panos.

Além do google e o stackoverflow, agora também temos inteligencias artificiais como ChatGPT entre outras para nos auxiliar com programação. Elas são excelentes ferramentas que vão explicar e ajudar a produzir códigos.

> ***Importante:*** É comum ver estudantes simplesmente copiando os códigos gerados pelas IA e colocando nos projetos. Se você não entender o código, ele pode quebrar partes do programa ou até mesmo adicionar vulnerabilidades no mesmo. Utilize inteligências artificiais como auxilio apenas.

Tipos de dados | Data Types

Existem diversos tipos de dados, e cada um é tratado de uma forma diferente por trás dos panos pela linguagem. A interação entre os dados vai depender do seu tipo.

Algumas linguagens são mais rigorosas e nos forçam a especificar o tipo de dado que cada variável vai receber. Estas são conhecidas como *Statically Typed Languages* (Linguagens estaticamente tipadas), e alguns exemplos são Java, C, C++, C#, Kotlin, Go.
Outras linguagens são menos exigentes e checam o tipo de dado quando o programa roda. Elas são conhecidas como *Dynamically Typed Languages* (Linguagens dinamicamente tipadas), e alguns exemplos são: Python, JavaScript, Ruby, PHP.

Nós já criamos variáveis e atribuímos dados a elas, assim:

```
message = "Hello, world"
```

Como podemos ver, nós não precisamos explicitamente dizer a python o tipo de dado que será atribuído à variável. Ele consegue descobrir o tipo sozinho. Mas em outras linguagens, como java, nós precisamos. Aqui está um exemplo de uma variável sendo criada em java, uma linguagem estaticamente tipada.

```
String message = "Hello, world";
```

Em java nós precisamos dizer que tipo de dado a variáveis deve esperar receber.

Mas se python consegue reconhecer o tipo de dado, por que preciso saber dos diferentes tipos? Cada tipo tem suas regras de uso e interação. Nós vamos aprender mais sobre isso ao falar de cada um deles a seguir.

String

Os dados do tipo `string` são caracteres alfanuméricos, ou seja, letras e números. Eles vêm entre aspas (duplas `"` ou únicas `'`).

> ***Nota:*** Quando uma palavra é digitada sem aspas, python a reconhece como um comando.

Vamos criar uma variável `name` e, desta vez, pediremos para o usuário digitar seu nome com a função `input`:

```
name = input("Qual o seu nome?")
print(name)
```

Ao rodarmos o código, veremos isto:

```
Qual o seu nome?João
João

Process finished with exit code 0
```

```
input()
```

É uma função que escreve na tela a mensagem entre parenteses, pega o que foi digitado pelo usuário e guarda em uma variável.

Ignorando caracteres

Como podemos ver, ao rodar o código, a resposta do usuário ficou colada com a pergunta. Isso pode ser resolvido com um espaço, mas podemos também fazer uma quebra de linha utilizando a barra inversa para utilizar caracteres especiais. Por exemplo:

```
name = input("Qual o seu nome?\n")
```

> ***Nota:*** `\n` no meio da string causa uma quebra de linha.

A barra inversa (também conhecida como *escape character*) também é utilizada para ignorar o caractere que vem logo em sua frente, como veremos a seguir.

Devo usar aspas duplas ou únicas?

Ambas funcionam da mesma forma: determinam onde a string começa e onde termina. Vamos supor que no meio da sua string você queira colocar uma palavra entre aspas:

```
"E então ele falou "eita""
```

Como pode ver, as cores estão até diferentes para sinalizar que python vai reclamar e levantar um erro. Ele não consegue entender que "eita" faz parte da string pois as primeiras aspas determinam onde a string começa e as segundas onde termina.

Uma possível solução é utilizar a barra inversa, como vimos anteriormente:

```
"E então ele falou \"eita\""
```

Neste caso, as cores não mudaram. python entende que as aspas ao redor de "eita" devem ser tratadas como string pois estão logo a frente de uma barra inversa.

Outra forma de lidar com a situação é utilizar as aspas únicas:

```
'E então ele falou "eita"'
```

Neste caso, as aspas únicas definem quando a string começa e quando termina, então as aspas duplas são tratadas como string.

Concatenação

Concatenação é o ato de juntar dados. Existem diversas formas de fazer isso em python, nós veremos algumas:

A forma mais comum de se concatenar é utilizando o operador `+`.

```
name = input("Qual o seu nome?\n")
print("Hello, " + name)
```

Neste caso, concatenamos a string `"Hello, "` e a string dentro da variável `name`, que foi digitada pelo usuário.

```
Qual o seu nome?
João
Hello, João

Process finished with exit code 0
```

Uma outra forma de concatenar variáveis é separando com virgula:

```
name = input("Qual o seu nome?\n")
print("Hello,", name)
```

> ***Nota:*** Neste caso, python coloca um espaço entre a variável.

Como você já deve estar imaginando, concatenar diversas variáveis em um texto gigante seria muito trabalhoso. Existe uma forma mais fácil de concatenar em python: utilizando uma `f string` ou *formatted string*.

```
name = input("Qual o seu nome?\n")
print(f"Hello, {name}")
```

Na `f string` todas as variáveis ficam dentro de chaves no meio da string. python entende que são variáveis e as substitui.

Exercício 1

1. Qual o resultado da seguinte expressão:

```
print("11" + "11")
```

2. Qual a diferença entre linguagens dinamicamente tipadas e linguagens estaticamente tipadas?

3. Madlibs é um jogo de substituição de palavras onde. após a criação de uma lista de palavras, elas são colocadas numa história. Crie um programa que peça ao usuário 1 nome e 2 adjetivos, então os coloque numa frase.

Números

Existem dois tipos de dados numéricos: integer (Int) e float (floating-point number). Integers são números inteiros, enquanto floats são números decimais.

Tipo de Dado	Exemplos
Int	-2, -1, 0, 1, 2, 3, 10, 45, 67, 106
Float	-1.32 , -1.0 , 0.0 , 1.57, 43.5, 100.0

Nós podemos fazer cálculos utilizando python, e para isso utilizamos operadores, assim como na matemática. Eles são:

Operador	Nome	O que faz
+	soma	soma os números
-	subtração	subtrai os números
*	multiplicação	multiplica os números
/	divisão	divide os números
%	módulo (modulus)	te dá o resto da divisão
**	exponenciação	eleva o número
//	divisor inteiro (floor division)	divide os números e té da um resultado arredondado para baixo

Alguns são bem simples e você deve se lembrar, outros nem tanto. Então vamos dar uma olhadinha em uns exemplos e esclarecer seus usos.

Módulo | Modulus

Este operador divide os números e o resultado é o resto. Qual será o resultado da expressão abaixo?

```
print(11 % 2)
```

isto mesmo! O resultado é 1.

```
1

Process finished with exit code 0
```

Ele divide 11 por 2 e restam 1.

Exponenciação

Caso não se lembre, exponenciação é o ato de multiplicar o número por ele mesmo uma ou mais vezes.

```
print(2 ** 3)
```

é o mesmo que 2^3 ou $2 \cdot 2 \cdot 2$, que é igual a 8.

```
8

Process finished with exit code 0
```

Divisor Inteiro | Floor Division

Com este operador, os números são divididos e python os arredonda o resultado para baixo.

```
print(10 // 3)
```

O resultado de 10 dividido por 3 é 3.333, porém, como utilizamos o divisor inteiro, python nos dá como resposta o número 3.

```
3

Process finished with exit code 0
```

Int e Float

Em determinadas linguagens, você só pode executar operações com dados do mesmo tipo. Em python este não é o caso. A linguagem tenta sempre entender o que você quer fazer e tenta trazer a resposta mais precisa possível.

Por conta disso, sempre que fazemos qualquer calculo entre um `int` e um `float`, o resultado vai ser um `float`. Python entende que a chance do resultado desta conta ser um número decimal é grande, então já faz a conversão pra `float`.

```
print(5.0 + 3)
```

```
8.0

Process finished with exit code 0
```

Em alguns casos, como na divisão, isto também ocorre. Mesmo que seja uma divisão entre dois números inteiros. Python sabe que existe uma boa chance de que o resultado de uma divisão seja um número com virgula.

```
print(4 / 2)
```

```
2.0

Process finished with exit code 0
```

Quantidade de casas decimais

As vezes python te dará um resultado com muitas casas decimais. Isto é normal e ocorre em muitas linguagens. Python sempre tenta entregar o valor mais próximo possível, o que é um pouco difícil devido a como computadores lidam com números.

```
print(0.2 + 0.1)
```

```
0.30000000000000004

Process finished with exit code 0
```

Mas e se eu quiser um número especifico de casas decimais? Neste caso, podemos formatar o resultado.

```
number = 1 + 0.9887984
print(f"Resultado não formatado: {number}")
print(f'Resultado formatado: {"%.2f" % number}')
```

aqui estamos somando um `int` e um `float`, então sabemos que o resultado será um `float` e python vai tentar deixá-lo o mais preciso possível. Na terceira linha, repare que a variável não foi sozinha. Nós passamos `"%.2f" % nome_da_variável`, que vai pegar o valor passado na variável e arredondar pra duas casas decimais.

> ***Nota:*** O `"%.2f"` sempre deve vir entre aspas. O `2` é a quantidade de casas decimais que você quer.

e este será nosso resultado:

```
Resultado não formatado: 1.9887983999999999
Resultado formatado: 1.99

Process finished with exit code 0
```

Ordem de operações matemáticas

Assim como na matemática, em python a ordem das operações matemáticas também deverá ser respeitada.

```
number = 5 + (2 + 3) ** 2
```

primeiro fazemos o calculo entre parênteses, depois calculamos a potência e, por fim, somamos.

```
30

Process finished with exit code 0
```

Números com underline

Números muito grandes são um pouco difíceis de ler. Python resolve isso nos deixando usar underlines para separar os números, deixando-os mais legíveis.

```
big_number = 4_540_000_000
print(big_number)
```

```
4540000000

Process finished with exit code 0
```

Boolean

O boolean é um tipo de dado com apenas dois valores possíveis: True (verdadeiro) e False (falso).

```
true_bool = True
false_bool = False
```

Bem fácil, não é?

Eu sei que ele não parece tão relevante ainda, mas o boolean é extremamente importante e vamos ver alguns do seus usos logo.

Revisitando variáveis

Agora que nós já temos um conhecimento sólido em relação às variáveis e tipos de dados, nós podemos nos aprofundar um pouco mais.

Até o momento, nós apenas atribuímos um valor à variável e a usamos. Mas como o nome diz, ela é variável. O que significa que podemos alterar seu valor a qualquer momento.

```
number = 100
number = 10
print(number)
```

Neste exemplo, primeiro criamos uma variável `number` e atribuímos uma integer de valor 100 a ela. Nada de novo. Logo abaixo, modificamos o valor da variável para 10. Ao rodarmos o código, temos este resultado:

```
10

Process finished with exit code 0
```

Python lê o código de cima para baixo, lembra? Então primeiro ele cria a variável, depois a modifica e, por ultimo, imprime o valor atual dela na tela, que é 10.

Como python é uma linguagem dinamicamente tipada, nós podemos até modificar seu tipo para string e python entenderia.

```
number = 100
number = "hello"
print(number)
```

```
hello

Process finished with exit code 0
```

Mais operadores de atribuição

Vamos imaginar que temos uma variável `number` de valor 2 e queiramos somar 10 ao seu valor. Como faríamos isso?

```
number = 2
number = number + 10
print(number)
```

Parece um pouco confuso,né? Mas vamos passo a passo. Primeiro criamos a variável e atribuímos o valor 2. Depois, nós atribuímos à variável `number` o valor de `number` + 10, ou seja, o novo valor de `number` é seu valor atual (2) mais a integer 10. Resultando:

```
12

Process finished with exit code 0
```

Outra forma de escrever isso, seria:

```
number = 2
number += 10
print(number)
```

Alguns outros operadores de atribuição:

Operador	O que faz
+=	soma ao valor antigo e atribui o resultado à variável
-=	subtrai do valor antigo e atribui o resultado à variável
*=	multiplica com o valor antigo e atribui o resultado à variável
/=	divide o valor antigo e atribui o resultado à variável
%=	calcula o módulo e atribui o resto à variável
**=	eleva o número e atribui o resultado à variável

Type Casting

Type casting é o ato de mudar o tipo de uma variável.

Enquanto python nos permite trabalhar com integers e floats ao mesmo tempo, mesmo sendo tipos de dados diferentes, isto não ocorre quando se trata de strings e números. Se tentarmos, python levanta um erro.

```
print("10" + 10)
```

```
Traceback (most recent call last):
  File "/home/joao/PycharmProjects/test_project/main.py", line 2, in <module>
    print("10" + 10)
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```

Python está reclamando e dizendo que pode apenas concatenar `string` (str) com `string`, e não `int`.

Input do usuário | User input

Quando usamos a função `input()` para pegar alguma informação do usuário, este dado sempre vem em formato de `string`, mesmo que o usuário tenha digitado um número. Para lidar com esta situação, precisamos converter os dados. E como fazemos isso?

Existem algumas funções que podemos usar:

```
int(dado)
```

transforma o dado passado em integer.

```
float(dado)
```

transforma o dado em float.

```
str(dado)
```

transforma o dado em string.

Um exemplo de uso:

```
number = input("digite um número:\n")
number = int(number)
```

ou, caso queira deixar o código ainda mais conciso:

```
number = int(input("digite um número:\n"))
```

Exercício 2

1. Crie um programa que peça para o usuário digitar 2 números e mostre sua soma.
2. Crie um programa que peça ao usuário seu nome e o ano em que nasceu, e mostre uma frase com seu nome e quantos anos ele tem/fará neste ano.
3. Crie um programa que peça ao usuário por dois números e mostre o valor da sua divisão com 3 casas decimais.

Constantes | Constants

Assim como as variáveis, as constantes são espacinhos na memória nos quais dados são armazenados. A diferença é que um valor atribuído a uma constante não pode ser alterado enquanto o programa está rodando.

Quando você tem certeza de que o valor de um dado não será alterado durante a execução do programa, pode escrevê-lo como uma constante. Desta forma, a linguagem saberá que

aquele valor não mudará e, por trás dos panos, fará as otimizações necessárias para funcionar de maneira mais eficiente.

Tá, mas como faço isto em python? Então, python não tem o tipo de dado "constante". Porém, como este dado existe em outras linguagens, nós seguimos a mesma nomenclatura para deixar claro que a variável deve ser tratada como uma constante, colocando seu nome em letras maiúsculas. Por exemplo:

```
CHARACTER_NAME = Lonk
```

Comentários | Comments

Comentários são linhas de códigos ignoradas por python, que escrevemos para nós mesmos ou para outros programadores. Espera, isso soa confuso. Deixa eu explicar melhor com um exemplo:

Imagine que você trabalhou em um projeto há uns anos e agora quer melhorá-lo. Ao abrir o projeto, você pode ler todo aquele código e decifrar o que cada linha está fazendo, mas seria muito mais rápido e fácil se houvessem comentários no código explicando o que cada bloco do código faz, ou explicando como você resolveu determinado problema.

Comentários são essenciais e facilitam muito a nossas vidas. Quando trabalhamos em equipe, eles nos permitem esclarecer para outros programadores o que nosso código está fazendo; Quando revisitamos nossos antigos projetos, nos permitem entender o que fizemos e como os fizemos.

Em python, tudo que é escrito na frente de um `#` é tratado como um comentário.

```
# Isto é um comentário
# E isto também!
```

Nós podemos também usar aspas triplas para fazer comentários de diversas linhas.

```
"""
Isto é um comentário
E isto também!

Tudo que é escrito aqui dentro é tratado como comentário.
"""
```

O Zen de Python | The Zen of Python

O Zen de Python é um texto escrito por Tim Peters que descreve a filosofia seguida pela comunidade para escrever um bom código em python.

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Eu comentarei apenas sobre alguns dos princípios. Alguns podem não fazer sentido agora mas farão posteriormente.

```
Beautiful is better than ugly.
# Bonito é melhor do que feio.

Explicit is better than implicit.
# Explícito é melhor do que implícito

Simple is better than complex.
# Simples é melhor do que complexo.

Readability counts.
# Legibilidade conta
```

Quando estamos começando a programar, não sabemos o que é bom e o que é ruim. Hoje vemos muitos vídeos nas redes sociais nos dizendo para escrever códigos de determinada maneira pois "é mais profissional", quando, na verdade, você só está deixando o código mais difícil de ler sem necessidade. Um código legível, bonito, de fácil entendimento, e simples, é sempre preferível e mais profissional do que o oposto.

E, por fim, talvez o princípio mais importante:

```
Now is better than never.
# Agora é melhor do que nunca.
```

Muitas vezes queremos aprender o máximo possível antes de começar. Sentimos que não estamos preparados, que não sabemos o suficiente. Porém, você não precisa saber de tudo para dar o primeiro passo. Mais importante do que escrever um código perfeito, é escrever um código que funciona. Mais pra frente, caso queira, você pode voltar e melhorá-lo.

Não deixe pra depois. Faça agora o que você pode com o que você tem.
