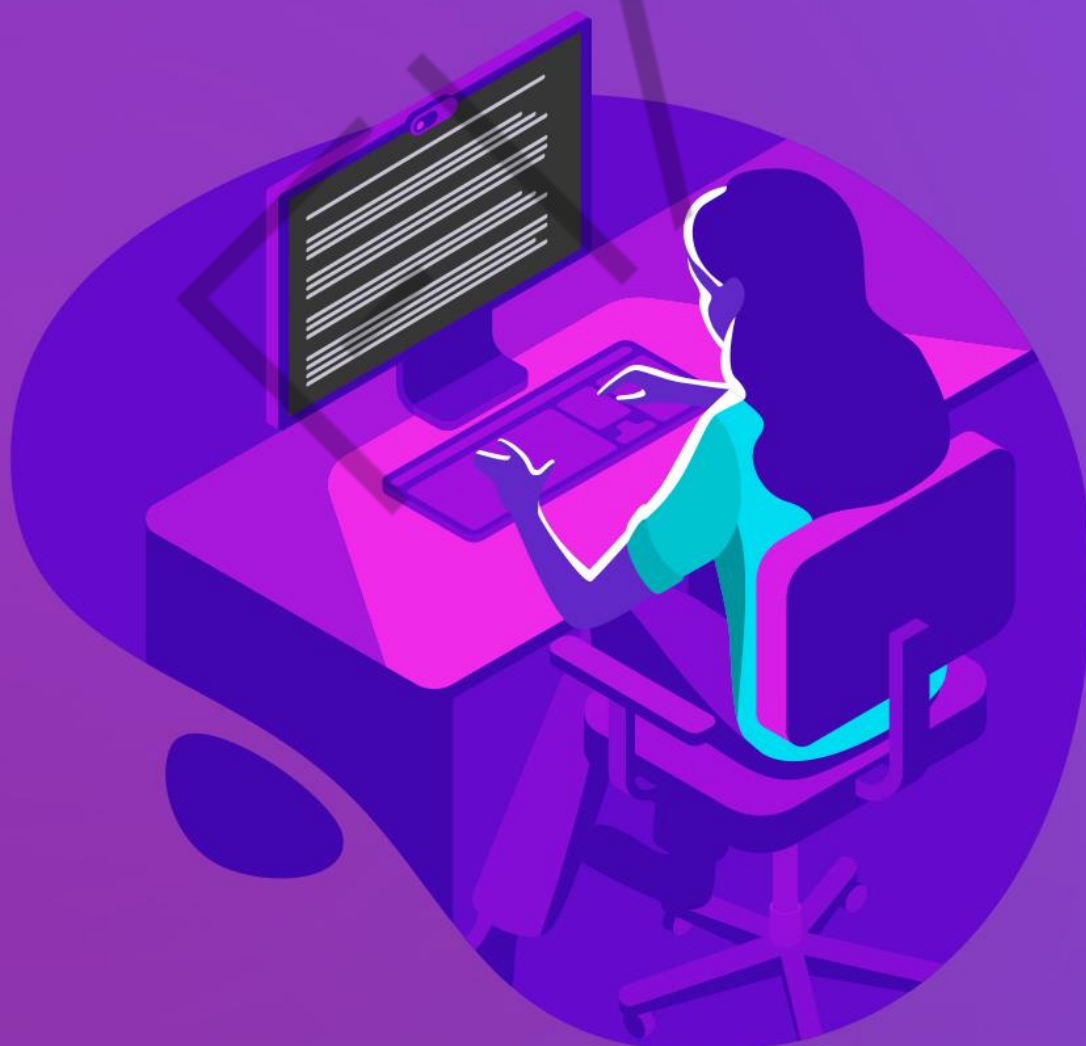


DEVELOPMENT ENVIRONMENT

VAMOS APRENDER A PROGRAMAR?

ANDRÉ DE FREITAS DAVID



10

LISTA DE FIGURAS

Figura 1 – Fluxograma representando a soma entre dois valores e sua exibição.....	9
Figura 2 – Versões do Python disponíveis	12
Figura 3 – Tela do instalador do Python.....	13
Figura 4 – Comando <i>python --version</i> no terminal do Windows	13
Figura 5 – Execução do Python através do terminal	14
Figura 6 – Salvando arquivo <i>.py</i> no Bloco de Notas.....	16
Figura 7 – Pasta atual no terminal é “C:\Users\andre”	17
Figura 8 – Pasta atual após o comando <i>cd..</i> é “C:\Users”	17
Figura 9 – Navegação até a pasta onde foi salvo o arquivo <i>.py</i>	17
Figura 10 – Execução de um script <i>.py</i>	18
Figura 11 – Tela ao abrir o PyCharm	19
Figura 12 – O projeto criado recebeu o nome de PrimeiroProjeto	20
Figura 13 – No projeto criado, será criado um novo arquivo Python.....	21
Figura 14 – Executando o script dentro do PyCharm.....	22
Figura 15 – Terminal aberto pelo PyCharm para executar o script	22
Figura 16 – Fluxograma para o algoritmo de calculadora	29
Figura 17 – Execução da soma em Python.....	30

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Pseudocódigo representando a soma entre dois valores e sua exibição	9
Código-fonte 2 – Programa em Python implementando o algoritmo	10
Código-fonte 3 – Exibição da mensagem “Olá, mundo!”, em Python.....	14
Código-fonte 4 – Criação e exibição da variável chamada <i>nome</i> com o conteúdo “Kevin Mitnick”.....	15
Código-fonte 5 – Captura de dados e exibição da variável <i>nome</i>	15
Código-fonte 6 – Script em Python para exibir o nome completo do usuário	21
Código-fonte 7 – Utilização de operadores de atribuição	23
Código-fonte 8 – Utilização dos operadores aritméticos	24
Código-fonte 9 – Utilização dos operadores relacionais	25
Código-fonte 10 – Utilização de operadores lógicos	25
Código-fonte 11 – Utilização de operadores de associação	27
Código-fonte 12 – Utilização de operadores de identidade	28
Código-fonte 13 – Pseudocódigo para o algoritmo de calculadora	30
Código-fonte 14 – Código em Python para implementar parte do algoritmo de calculadora	30
Código-fonte 15 – Uso do comando <i>type</i> para verificar o tipo das variáveis.....	31
Código-fonte 16 – Conversão de tipos de dados em Python	32
Código-fonte 17 – Calculadora totalmente implementada em Python.....	32
Código-fonte 18 – Calculadora totalmente implementada em Python.....	33
Código-fonte 19 – Calculadora totalmente implementada.....	34
Código-fonte 20 – Calculadora totalmente implementada.....	34

LISTA DE QUADROS

Quadro 1 – Operadores de atribuição	23
Quadro 2 – Operadores aritméticos	24
Quadro 3 – Operadores relacionais	24
Quadro 4 – Operadores lógicos	25
Quadro 5 – Operadores de associação.....	27
Quadro 6 – Operadores de identidade	27
Quadro 7 – Precedência dos operadores.....	28
Quadro 8 – Tipos de dados básicos.....	31

LISTA DE TABELAS

Tabela 1 – Tabela verdade operador not (não)	26
Tabela 2 – Tabela verdade operador and (e)	26
Tabela 3 – Tabela verdade operador or (ou).....	26

EMSE

SUMÁRIO

1 VAMOS APRENDER A PROGRAMAR?	7
1.1 Que negócio é esse de programação?	7
1.2 A famosa palavra “algoritmo”	7
1.3 Todo algoritmo é um programa de computador?	8
1.4 Aprendendo com a mão na massa!	10
1.4.1 Etapa 1: Instalando o Python.	11
1.4.2 Etapa 2: Conhecendo alguns comandos.	14
1.4.3 Etapa 3: Esqueça a tela preta!	18
2 OS OPERADORES	23
2.1 Operadores de atribuição	23
2.2 Operadores aritméticos	24
2.3 Operadores relacionais	24
2.4 Operadores lógicos	25
2.5 Operadores de associação	27
2.6 Operadores de identidade	27
2.7 Precedência dos operadores	28
2.8 Hora de exercitar	29
2.8.1 A calculadora!	29
3 O PATINETE ELÉTRICO!	33
3.1 Vamos trocar?	34
REFERÊNCIAS	36

1 VAMOS APRENDER A PROGRAMAR?

1.1 Que negócio é esse de programação?

Se você já navegou em páginas e fóruns de tecnologia na web, com certeza, já se deparou com frases como: “Instala esse programa aqui que ele resolve!”, “Qual linguagem de programação devo estudar?” ou “Para ser um programador, preciso conhecer matemática?”.

Amada por muitos, temida por outros e ignorada por boa parte dos usuários comuns, a programação é uma das áreas de maior importância para a tecnologia como a conhecemos hoje!

Se um *hardware* muito avançado tem alguma utilidade, é porque alguém criou um *algoritmo* que fosse capaz de utilizá-lo para cumprir alguma tarefa.

Ao longo deste curso, vamos aprender a *compreender o mundo e propor soluções* pensando como programadores! Vamos lá?

1.2 A famosa palavra “algoritmo”

Você já deve ter perdido as contas de quantas vezes ouviu alguém dizer que o *algoritmo* do Facebook esconde determinados conteúdos, ou de que o *algoritmo* do Google sabe tudo sobre você, mas o que é esse tal de algoritmo?

Se escolhermos ir por um caminho etimológico, descobriremos que a palavra “algoritmo” deriva do nome de um matemático persa do século IX, Al-Khwarizmi. A esse matemático são atribuídas as construções dos primeiros processos para realização de operações aritméticas.

Outro matemático famosíssimo a criar um algoritmo que você usou na escola é Euclides, responsável por estruturar o processo para o cálculo do Máximo Divisor Comum (o MDC).

Pode ser que você não esteja vestido de Indiana Jones e queira uma definição mais prática e menos histórica, então, podemos dizer que algoritmo é uma

sequência ordenada de instruções que visa resolver um determinado problema. Sim, a *receita de bolo da vovó* é um algoritmo!

No entanto, será que todo algoritmo é um programa de computador?

1.3 Todo algoritmo é um programa de computador?

Se um algoritmo é qualquer sequência ordenada de instruções para atingir um objetivo, como uma receita de bolo ou as instruções de montagem de um móvel de madeira, então, é fácil chegar à conclusão de que algoritmos e programas não são sinônimos.

“Então, por que os termos estão sempre associados, professor?”, é o que você deve estar se perguntando agora.

Nós podemos (e faremos muito isso ao longo do curso) usar os computadores para criar programas que realizem as instruções de um algoritmo, mas existem muitas formas de representar essa sequência de passos!

No meio da *computação*, duas das formas mais comuns de representação algorítmica são os *fluxogramas* (também conhecidos como diagramas de blocos) e os *pseudocódigos* (que podem ter diferentes versões, como o Portugol).

Vamos imaginar um *algoritmo* que recebe 2 valores do usuário e realiza a soma entre eles. Ao final, devemos exibir o resultado dessa soma.

A representação através de um *fluxograma* contém uma sequência de blocos geométricos indicando a ordem em que os eventos ocorrerão. Já a representação através de um *pseudocódigo* é feita através de texto, com uma linguagem que não é uma linguagem de programação, mas que apresenta uma estrutura formal.

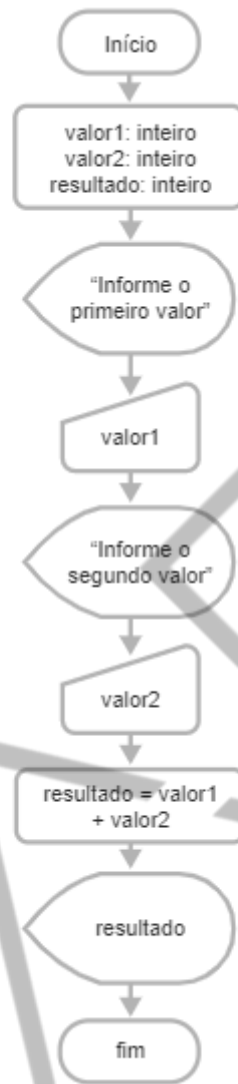


Figura 1 – Fluxograma representando a soma entre dois valores e sua exibição
Fonte: Elaborado pelo autor (2019)

```
algoritmo "Soma"
variáveis
    valor1, valor2, resultado: inteiro
início
    Escreva "Digite o primeiro valor"
    Leia valor1
    Escreva "Digite o segundo valor"
    Leia valor2
    resultado = valor1 + valor2
    Escreva resultado
Fim
```

Código-fonte 1 – Pseudocódigo representando a soma entre dois valores e sua exibição
Fonte: Elaborado pelo autor (2019)

Podemos observar um mesmo algoritmo representado em duas formas distintas (*fluxograma* e *pseudocódigo*) e nenhum programa de computador foi escrito até esse momento.

Esse mesmo algoritmo pode ser *implementado* usando uma linguagem de programação qualquer, como Python, Java, C++, PHP etc.

```
valor1 = input("Digite o primeiro valor ")
valor2 = input("Digite o segundo valor ")
resultado = int(valor1) + int(valor2)
print(resultado)
```

Código-fonte 2 – Programa em Python implementando o algoritmo
de soma entre dois valores e sua exibição
Fonte: Elaborado pelo autor (2019)

Perceba ainda que, embora as palavras ou símbolos usados nos algoritmos e na implementação final sejam diferentes, é possível observar facilmente que cumprem a mesma tarefa.

Que tal começarmos a criar nossos próprios algoritmos e programas imediatamente?

1.4 Aprendendo com a mão na massa!

Uma das melhores formas de aprendermos algo é colocando a mão na massa, não é mesmo?

Por essa razão, o nosso estudo da lógica de programação, do pensamento algorítmico e das estruturas básicas de programação de computadores se dará com uma linguagem de programação que ganha cada vez mais espaço no mercado: o Python!

A todo momento, faremos comparações entre os programas que desenvolvermos em Python com alguma representação de algoritmos, como o *fluxograma*.

Se começarmos a programar é nosso objetivo final, vamos criar um algoritmo para ele?

1.4.1 Etapa 1: Instalando o Python

Para escrevermos um algoritmo, basta usarmos um pedaço de papel e um lápis. Um papel de pão e um pedaço de carvão serviriam, afinal, já que os algoritmos são apenas as sequências lógicas para atingir um objetivo, não precisamos de nenhum dispositivo específico para representá-los.

Como faremos nosso estudo diretamente com uma linguagem de programação, precisamos de um ambiente que seja capaz de nos auxiliar. Esse ambiente é composto de ao menos duas partes: a *IDE* e o *interpretador* (ou compilador, dependendo da linguagem).

Provavelmente, você se pergunte: quando escrevemos um código em linguagem de programação, como o computador sabe o que ele tem que fazer? Quem faz a função de traduzir esse código em algo que o computador seja capaz de executar é o interpretador ou o compilador.

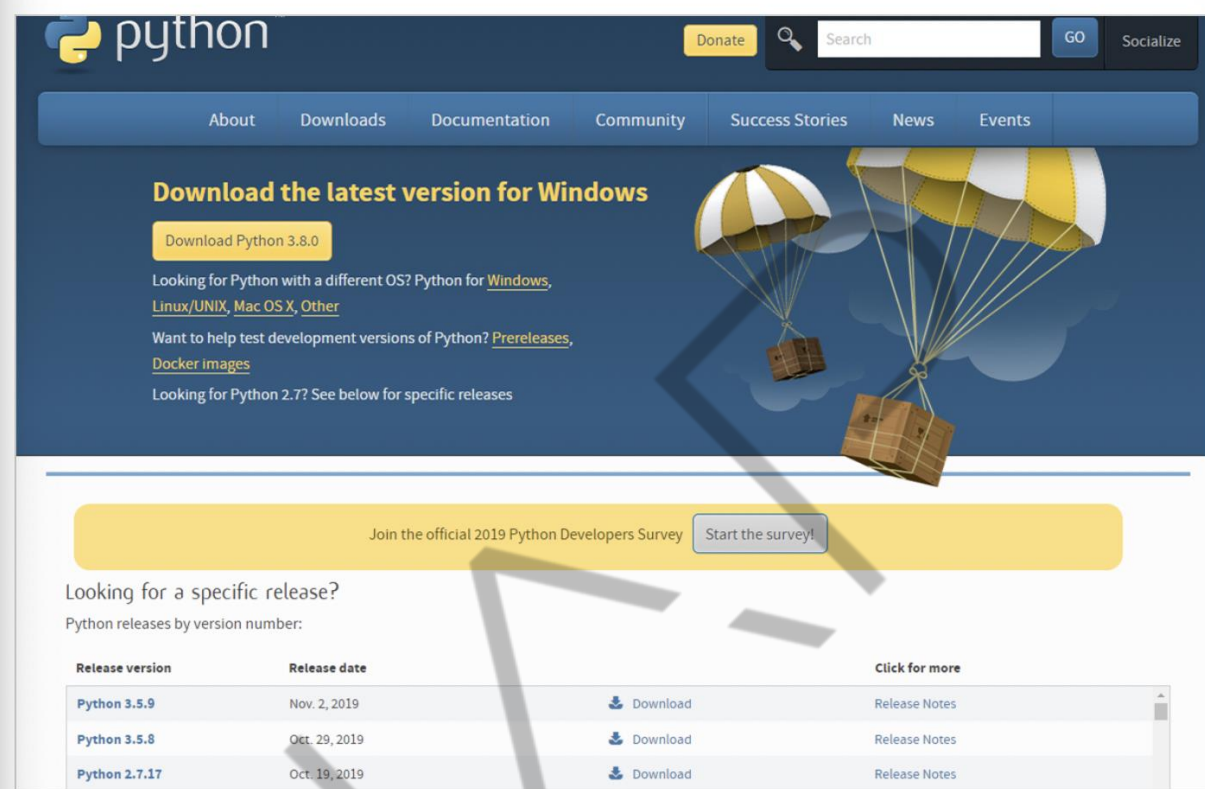
Quando uma linguagem é compilada, todo o código é transformado em um arquivo executável que está em código de máquina e pode ser executado por ela. Dessa forma, precisamos gerar um código de máquina para cada arquitetura em que quisermos rodar nosso software: um para Windows, outro para MacOS, outro para Linux e assim por diante.

Quando uma linguagem é interpretada, o código original não é transformado em código de máquina, mas, sim, interpretado, de forma que qualquer máquina que tiver um interpretador instalado conseguirá rodar o código original.

No caso do Python, existem atualmente duas versões da linguagem e, consequentemente, dois interpretadores: o Python 2.x e o Python 3.x. Como existem pequenas diferenças entre ambas e o nosso foco aqui é no desenvolvimento da lógica de programação e não nas minúcias da linguagem, nosso curso se baseará na versão 3.

Quando se trabalha com desenvolvimento de sistemas, é importante estar disposto a encarar mudanças nos procedimentos de instalação das ferramentas, mas o roteiro geral acaba sendo o mesmo: acessar o site da ferramenta e realizar o download e a instalação.

No caso do Python, o site do projeto é o <<https://www.python.org/>>, que conta com um link para a página de downloads, reunindo todas as versões disponíveis:



Release version	Release date	Click for more
Python 3.5.9	Nov. 2, 2019	Download Release Notes
Python 3.5.8	Oct. 29, 2019	Download Release Notes
Python 2.7.17	Oct. 19, 2019	Download Release Notes

Figura 2 – Versões do Python disponíveis
Fonte: Python (2021)

Uma dica valiosa: a maior parte das ferramentas de programação conta com atualizações constantes. Dessa forma, a não ser que você tenha certeza que deseja uma versão específica da ferramenta, usar a versão apontada como *latest* (última) é sempre uma boa pedida.

Após fazer o download da última versão do Python 3, a execução do instalador é bem simples, sendo apenas necessário ficar atento à opção de “adicionar ao path”, ou equivalente. Isso significa que se você digitar *python* no seu terminal, o sistema operacional reconhecerá o local de instalação do interpretador.

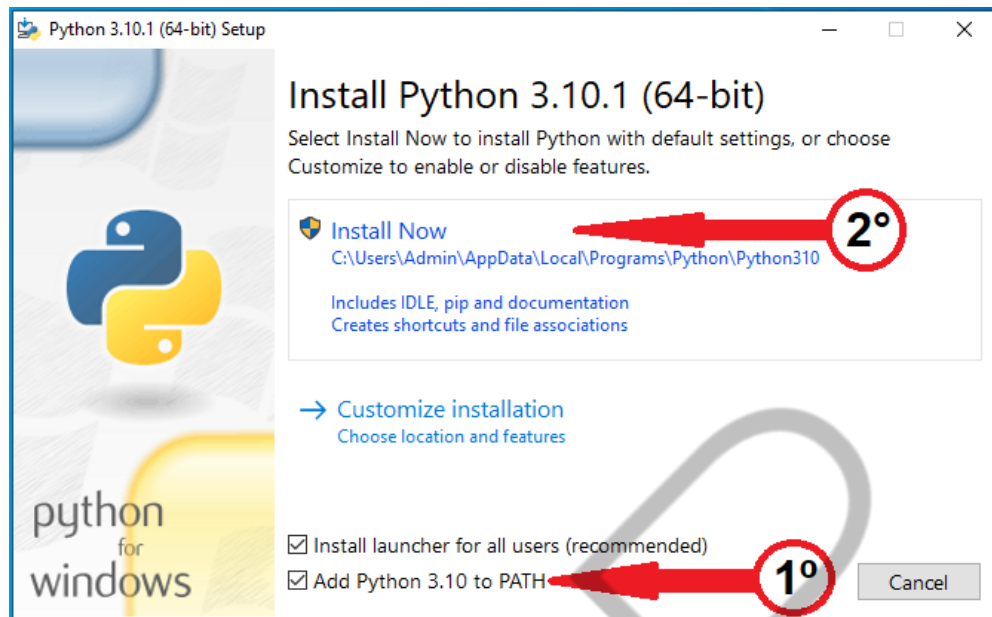


Figura 3 – Tela do instalador do Python
Fonte: Elaborado pelo autor (2021)

Para saber se tudo correu bem, basta abrir o terminal de comando do seu sistema operacional (no caso do Windows, é o famoso *cmd*) e digitar a instrução *python --version*. Se tudo tiver corrido bem, o terminal deverá mostrar a versão do Python que está instalada na sua máquina.

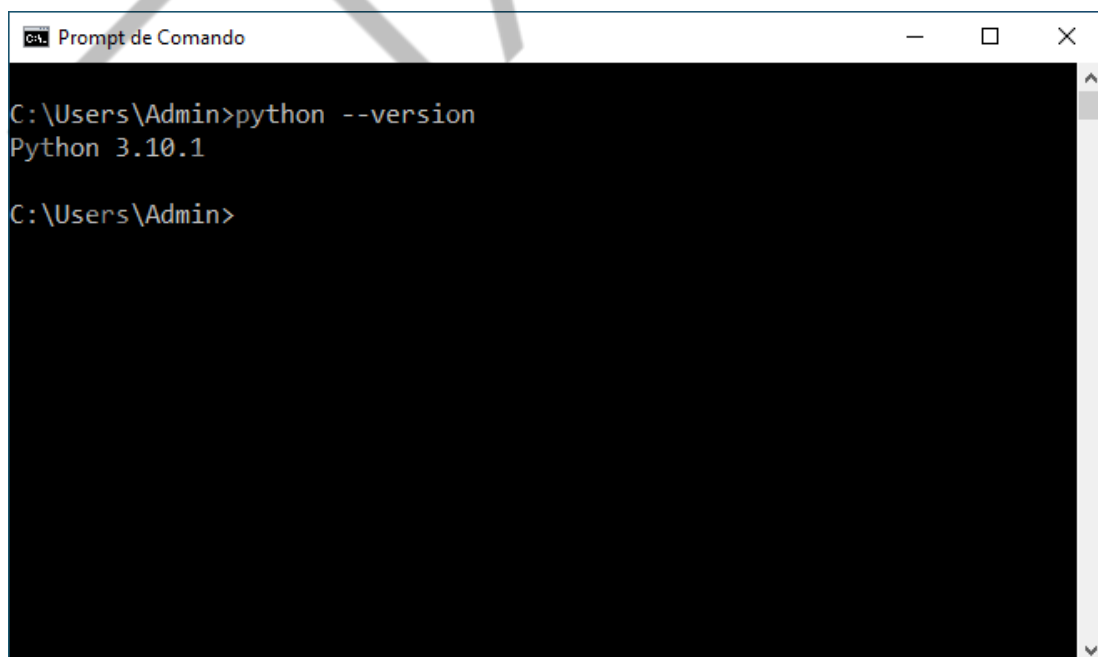


Figura 4 – Comando *python --version* no terminal do Windows
Fonte: Elaborado pelo autor (2021)

Agora que já sabemos que o Python foi instalado corretamente, que tal testarmos algumas instruções de programação?

1.4.2 Etapa 2: Conhecendo alguns comandos.

Toda linguagem de programação possui algumas *palavras reservadas* que têm algum significado dentro daquela linguagem de programação. Essas palavras reservadas podem ser comandos, por exemplo, que realizam tarefas específicas.

Para exibir uma mensagem na tela do computador, usaremos o comando *print()*. A sintaxe desse comando nos obriga a fornecer, entre parênteses, o texto que queremos exibir na tela.

Para testarmos esse primeiro comando, podemos abrir o terminal do nosso sistema operacional e digitar a palavra *python*, para que o interpretador seja aberto:

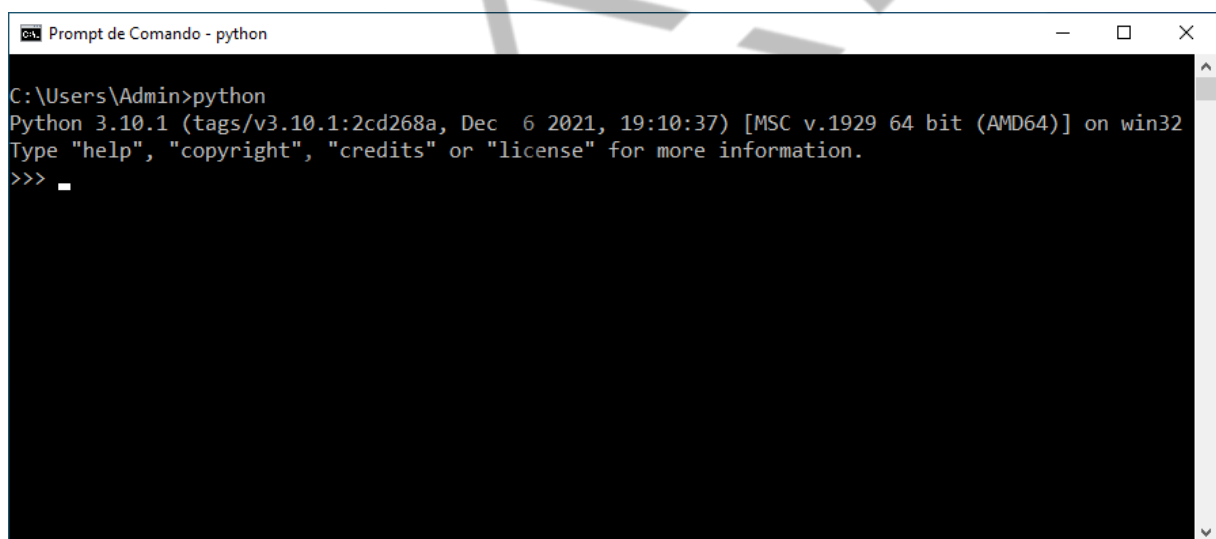


Figura 5 – Execução do Python através do terminal
Fonte: Elaborado pelo autor (2021)

Com o interpretador do Python em execução, tudo o que digitarmos nessa linguagem será imediatamente interpretado e executado após teclarmos ENTER. Vamos tentar executar o seguinte comando, pressionando ENTER ao final:

```
print("Olá, mundo!")
```

Código-fonte 3 – Exibição da mensagem "Olá, mundo!", em Python
Fonte: Elaborado pelo autor (2019)

Se tudo correu bem, após teclar ENTER, a mensagem “Olá, mundo!” deve ter sido exibida na tela.

E se quisermos exibir uma mensagem personalizada com o nome do usuário?

Para isso, entra em cena um conceito importantíssimo de programação: a variável.

Variável é um espaço que um programa pode reservar na memória RAM do computador para armazenar temporariamente alguns dados, como algo que o usuário tenha digitado ou o resultado de um cálculo.

Outras linguagens de programação possuem comandos específicos para a criação de variáveis, mas a linguagem Python entende que uma palavra que seja escrita do lado esquerdo do sinal de = (igual) é uma variável e cria automaticamente caso seja seu primeiro uso. Então, se escrevermos os comandos a seguir (lembrando de teclar ENTER ao final de cada linha), criaremos uma variável com o nome do usuário e exibiremos na tela:

```
nome = "Kevin Mitnick"  
print(nome + " é um programador incrível!")
```

Código-fonte 4 – Criação e exibição da variável chamada *nome* com o conteúdo “Kevin Mitnick”
Fonte: Elaborado pelo autor (2019)

Infelizmente, no programa anterior, o código não está recebendo o nome do usuário, apenas exibindo um nome fixo. Para permitir aos usuários digitarem informações dentro de variáveis, devemos utilizar o comando *input()*.

Em sua sintaxe, o comando *input()* exige a presença de uma variável antes do comando e de uma mensagem de texto entre parênteses: *nome_da_variavel = input("Mensagem de texto :")*.

Reescrevendo o programa anterior para aceitar que o usuário digite seu próprio nome, teremos o seguinte:

```
nome = input("Por favor, digite seu nome: ")  
print(nome + " é um programador incrível!")
```

Código-fonte 5 – Captura de dados e exibição da variável *nome*
Fonte: Elaborado pelo autor (2019)

Conseguimos criar três programas até agora, mas vamos combinar que esse negócio de escrever os comandos em uma tela de console e ter que executar um de cada vez não está com nada, não é? Vamos melhorar isso!

Para sairmos do interpretador do Python, basta escrevermos `exit()` e teclarmos *ENTER*.

Agora vamos reunir todas as linhas do nosso programa em um só arquivo, que damos o nome de *script*. Para fazer isso, basta abrir o Notepad (Bloco de Notas), ou o equivalente do seu sistema operacional.

No editor de texto aberto, vamos digitar o código do programa anterior, que pedia o nome do usuário e o exibia, e salvar o arquivo com o formato *.py*.

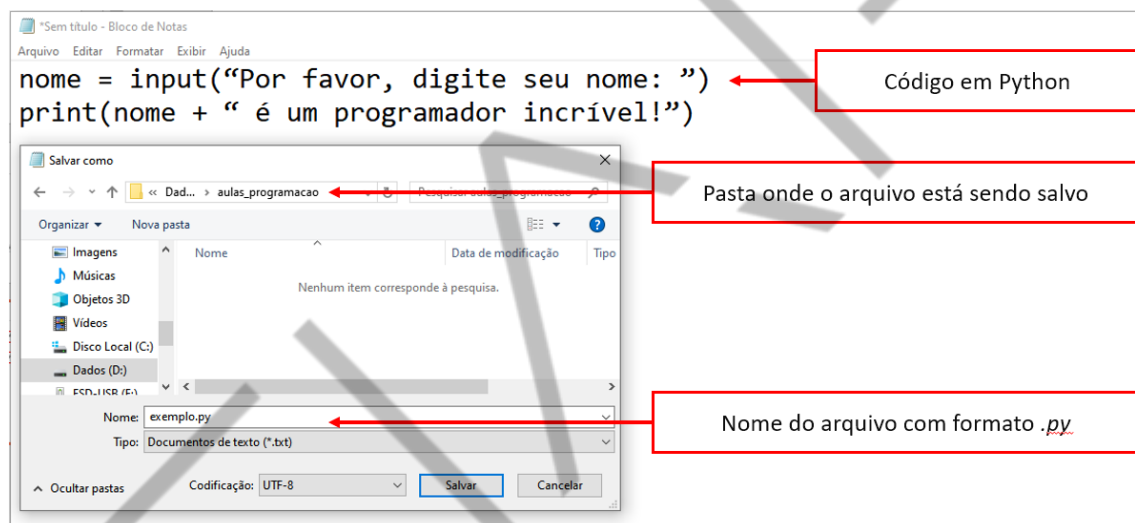


Figura 6 – Salvando arquivo *.py* no Bloco de Notas
Fonte: Elaborado pelo autor (2019)

Uma vez que o arquivo de texto contém uma série de comandos em Python, podemos executar esse script diretamente do terminal. Primeiramente, vamos navegar até a pasta que contém o script. Para fazer isso, basta usar o comando `cd` dentro do terminal, seguido do nome da pasta que queremos abrir ou seguido de dois pontos (..) para voltar para a pasta anterior.

Ficou confuso? Fique tranquilo que temos um exemplo supersimples. Quando olho para o meu terminal, ele indica em que pasta estou agora:

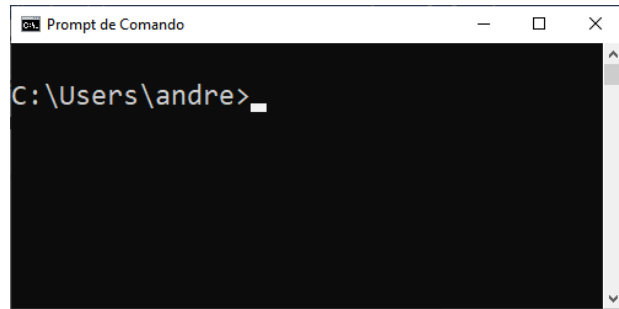


Figura 7 – Pasta atual no terminal é "C:\Users\andre"
Fonte: Elaborado pelo autor (2019)

Para voltar para a pasta anterior, posso digitar o comando "cd..".

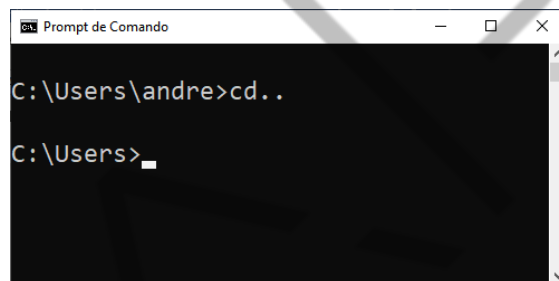


Figura 8 – Pasta atual após o comando cd.. é "C:\Users"
Fonte: Elaborado pelo autor (2019)

Como o arquivo de exemplo foi salvo na pasta "aulas_programação", vamos mostrar a navegação até lá, mas você deve ir até a pasta em que salvou o arquivo .py, ok?

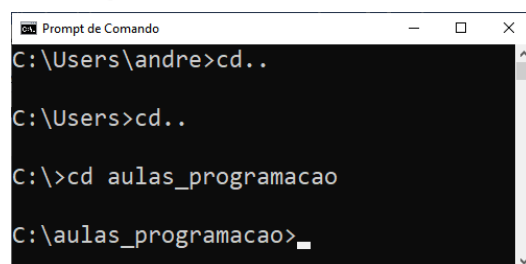


Figura 9 – Navegação até a pasta onde foi salvo o arquivo .py
Fonte: Elaborado pelo autor (2019)

Para executarmos um arquivo Python, basta escrevermos *python nome_do_arquivo.py*. No nosso caso, o comando será *python exemplo.py* e, voilà: nosso script está em execução!

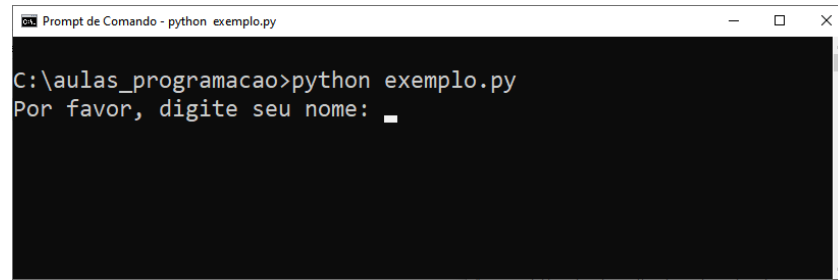


Figura 10 – Execução de um script .py
Fonte: Elaborado pelo autor (2019)

Por mais que alguns programadores gostem de escrever os scripts em Python em editores de texto simples, para quem está começando a programar é muito desconfortável usar o terminal para executar os programas. É chegada a hora de usarmos uma ferramenta mais adequada para a programação: a IDE!

1.4.3 Etapa 3: Esqueça a tela preta!

Agora que já sabemos como criar nossos programas em Python, que tal usarmos uma ferramenta mais adequada para isso do que o bloco de notas?

As IDEs (*Integrated Development Environment*) são ferramentas usadas para escrever programas em uma linguagem de programação específica. Elas não contêm apenas os editores de texto, mas ferramentas que permitem verificar erros no código, executar o programa diretamente no interpretador ou compilador e, em alguns casos, montar interfaces gráficas.

Cada linguagem de programação possui diversas IDEs, algumas delas oficiais e outras feitas pela comunidade. Toda IDE possui vantagens e desvantagens e não há grande valor em entrar em debates acalorados sobre qual é a melhor: cada programador vai encontrar uma IDE que lhe sirva bem.

Para os propósitos do nosso estudo, usaremos o PyCharm, mas você pode testar outras e utilizar a que mais lhe agrade.

Para instalar o PyCharm, basta baixar o instalador no site do projeto, <<https://www.jetbrains.com/pycharm/>>, ficando atento a apenas um detalhe: essa IDE possui a versão Professional, que é paga, e a versão Community, que é gratuita e nós a utilizaremos.

Durante a instalação, as opções-padrão fazem o necessário, então, podemos seguir com o famoso “next, next, next, finish”.

Quando o PyCharm estiver instalado e for executado, vamos selecionar a opção de criar um novo projeto. Tela ao abrir o PyCharm, pode-se escolher entre criar um novo projeto ou abrir um preexistente.



Figura 11 – Tela ao abrir o PyCharm
Fonte: Elaborado pelo autor (2021)

Ao selecionarmos a opção para criar um projeto, tudo o que precisaremos fazer é indicar o nome e o local onde o projeto ficará!

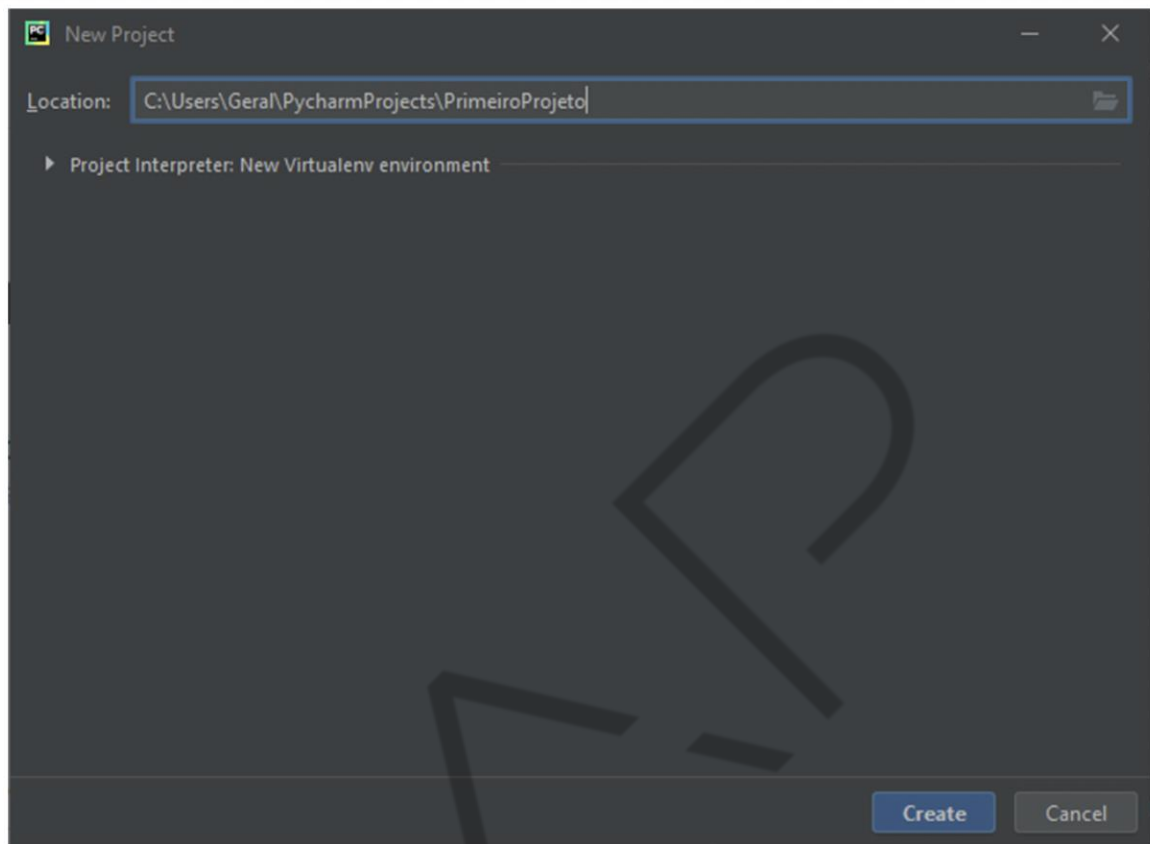


Figura 12 – O projeto criado recebeu o nome de PrimeiroProjeto
Fonte: Elaborado pelo autor (2021)

Uma vez que o projeto foi criado, podemos, finalmente, criar nosso primeiro script dentro da IDE. Para isso, nós devemos clicar sobre o nome do projeto com o botão direito do mouse, escolhendo as opções *New* e *Python File*:

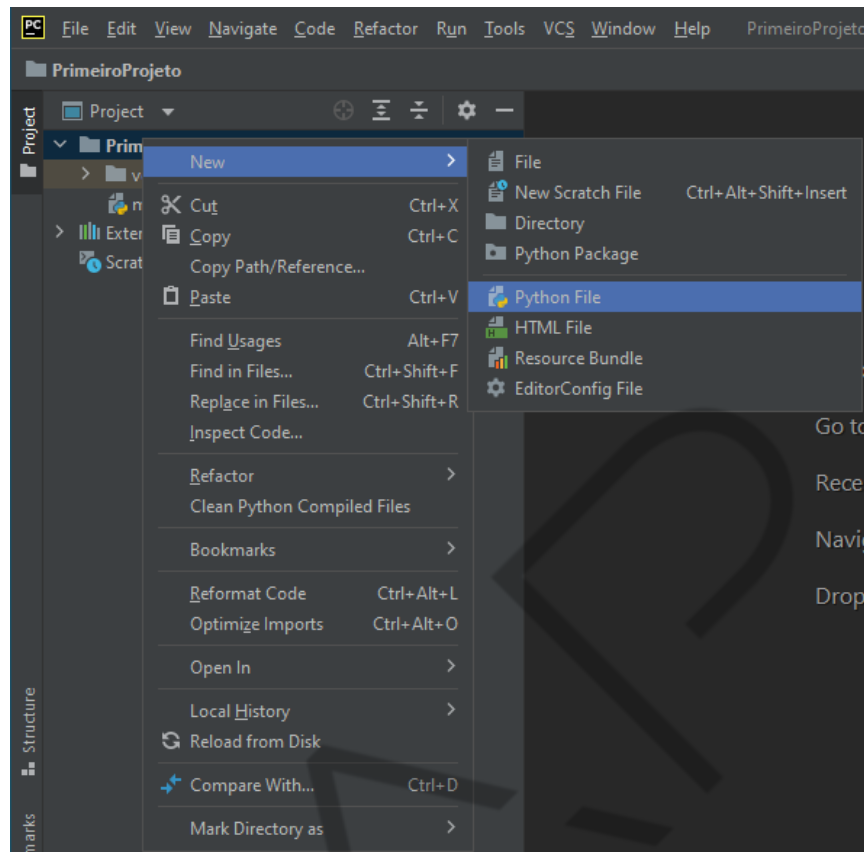


Figura 13 – No projeto criado, será criado um novo arquivo Python
Fonte: Elaborado pelo autor (2021)

Vamos chamar o novo arquivo de *nome_sobrenome.py* e, dentro dele, escrever um script em Python que implementa um algoritmo receber o nome e o sobrenome de um usuário, exibindo o nome completo em sequência.

```
print("Esse programa exibirá seu nome completo. ")
nome = input("Digite seu primeiro nome, por favor: ")
sobrenome = input("Digite seu sobrenome, por favor: ")
nome_completo = nome + " " + sobrenome
print(nome_completo)
```

Código-fonte 6 – Script em Python para exibir o nome completo do usuário
Fonte: Elaborado pelo autor (2019)

Para executarmos o script que criamos, basta clicarmos sobre o nome do arquivo com o botão direito do mouse e selecionarmos a opção *Run nome_sobrenome.py* e o próprio PyCharm abrirá um terminal para executá-lo.

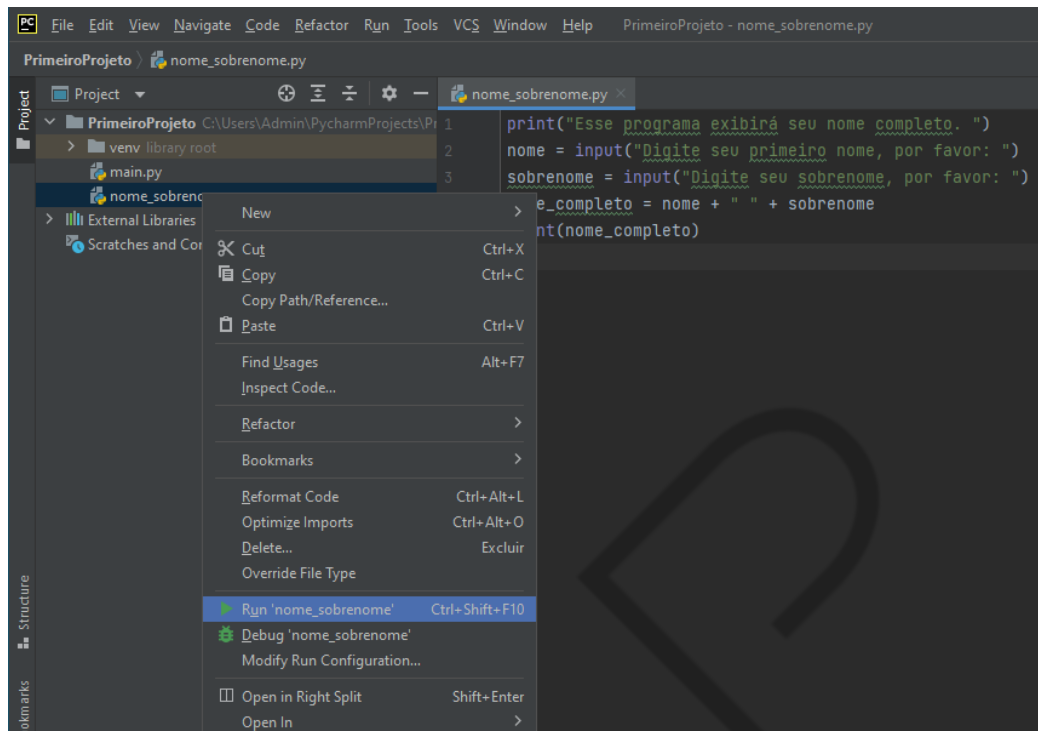


Figura 14 – Executando o script dentro do PyCharm
Fonte: Elaborado pelo autor (2021)

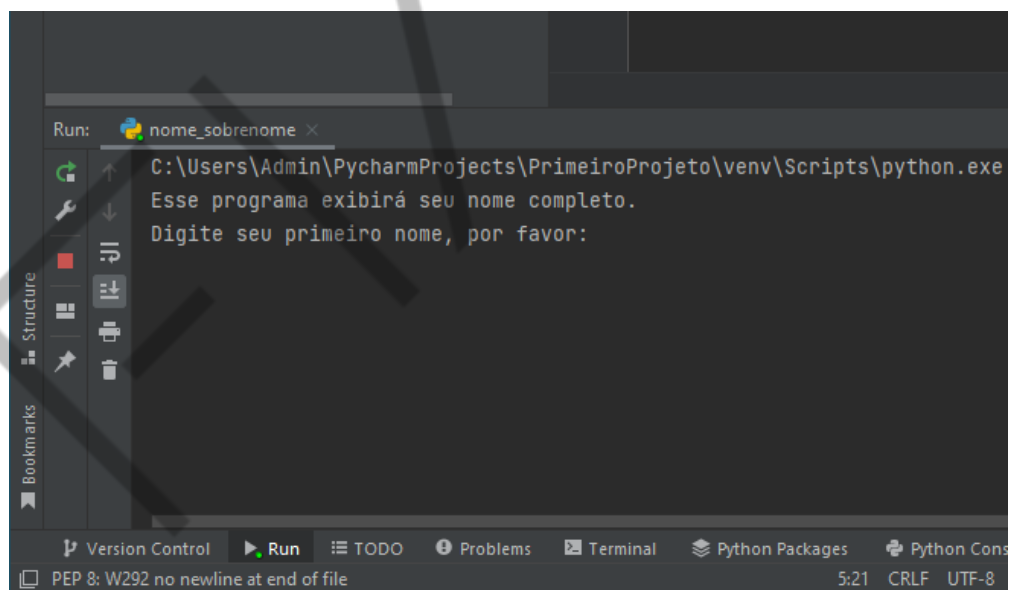


Figura 15 – Terminal aberto pelo PyCharm para executar o script
Fonte: Elaborado pelo autor (2021)

Sendo muito mais fácil executar os scripts dentro do PyCharm do que navegando pelos diretórios do Terminal, vamos continuar usando essa ferramenta para fazer alguns exercícios.

DICA: Para encerrar um terminal no PyCharm, basta clicar no quadrado vermelho localizado ao lado dele.

2 OS OPERADORES

Vamos falar agora sobre os operadores que são símbolos especiais que utilizamos na linguagem de programação para representar uma operação. No Python, os principais operadores são os: de atribuição, aritméticos, relacionais, lógicos, de identidade e de associação.

2.1 Operadores de atribuição

Durante a programação, faremos diversas atribuições de valores às variáveis. Esta tarefa pode ser realizada com a utilização dos operadores de atribuição que servem para incluir/associar valores a uma variável/objeto.

Operador	Exemplo	Equivalência
=	x = 3	x = 3
+=	x += 1	x = x + 1
-=	x -= 1	x = x - 1
*=	x *= 2	x = x * 2
/=	x /= 2	x = x / 2
%=	x %= 2	x = x % 2

Quadro 1 – Operadores de atribuição
Fonte: Elaborado pelo autor (2021)

Na criação dos códigos, podemos criar comentários que não serão executados, mas que facilitam a compreensão e a organização do código. No código-fonte, podemos utilizar o sinal # (cerquilha) para adicionarmos um comentário. Para comentários com mais de uma linha, podemos utilizar as aspas triplas: `"""texto"""`.

```
#atribuição da soma de dois números inteiros
resultado = 2 + 3
print("Resultado:", resultado)
#atribuição de um texto (string)
nome = "Pedro"
print("Nome: ", nome)
nome2 = "Maria"
#saída f"string" com a variável entre as chaves
print(f"Olá, {nome2}. Tudo bem?")
#somando dois ao valor anterior da variável
resultado += 2
print("Resultado:", resultado)
```

Código-fonte 7 – Utilização de operadores de atribuição
Fonte: Elaborado pelo autor (2021)

2.2 Operadores aritméticos

São os operadores que utilizamos quando precisamos realizar os diversos cálculos aritméticos.

Operador	Operação	Precedência
+	Adição	Menor prioridade
-	Subtração	
*	Multiplicação	
/	Divisão	
//	Divisão inteira	
%	Resto da divisão ou módulo	
**	Exponenciação ou potenciação	Maior prioridade

Quadro 2 – Operadores aritméticos
Fonte: Elaborado pelo autor (2021)

Assim como na Matemática, os operadores aritméticos possuem precedência (prioridade de execução). Quando utilizamos operadores aritméticos de mesma prioridade, as operações são realizadas da esquerda para a direita. Obs.: os parênteses podem ser utilizados para priorizar uma operação.

A expressão $2 + 3 / 2$ resulta em 2.5 ou 3.5? Vamos testar!

```
resultado = 2 + 3 / 2
print("Resultado: ", resultado)
```

Código-fonte 8 – Utilização dos operadores aritméticos
Fonte: Elaborado pelo autor (2021)

IMPORTANTE: A linguagem utiliza o ponto e não a vírgula para representar a parte fracionária de um número do tipo real: 3.5 ao invés de 3,5.

2.3 Operadores relacionais

Quando precisamos realizar comparações, é a hora de utilizarmos os operadores relacionais ou de comparações.

Operador	Função	Exemplo
==	Igual	$5 == 5$
!=	Diferente	$5 != 5$
>	Maior que	$5 > 5$
<	Menor que	$5 < 5$
>=	Maior igual	$5 >= 5$
<=	Menor igual	$5 <= 5$

Quadro 3 – Operadores relacionais
Fonte: Elaborado pelo autor (2021)

Podemos utilizá-los na linguagem para verificarmos se um número/objeto é igual, diferente, maior ou menor que outro. O resultado de uma comparação será True (verdadeiro) ou False (falso). Na linguagem Python, utilizamos as palavras-chaves True e False, que são objetos que armazenam, respectivamente, os valores 1 e 0.

```
print(1 == 5)
print(1 != 5)
print(1 > 2)
print(1 < 2)
print(1 >= 2)
print(2 <= 2)
print(1 == True)
print(0 == False)
```

Código-fonte 9 – Utilização dos operadores relacionais
Fonte: Elaborado pelo autor (2021)

2.4 Operadores lógicos

Em algumas situações, precisamos trabalhar com valores lógicos, conhecidos também como valores booleanos. Para isso, utilizamos os operadores lógicos que podem agrupar as operações com lógica booleana. Cada operador lógico obedece a algumas regras de entrada e saída.

Operador	Exemplo	Equivalência
or	A or B	ou
and	A and B	e
not	not A	não

Quadro 4 – Operadores lógicos
Fonte: Elaborado pelo autor (2021)

Podemos utilizar este tipo de operador em expressões lógicas mais complexas, combinando-os com os operadores relacionais.

```
A = 1 == 2
B = 2 > 3
print(A)
print(B)
#Operações
print(A and B)
print(A or B)
print(not A)
print(not B)
print(8>=5 and 8<=10)
```

Código-fonte 10 – Utilização de operadores lógicos

Fonte: Elaborado pelo autor (2021)

O operador lógico **not** (não) também é conhecido como inversor e representa a negação do valor de entrada, ou seja, o inverso da entrada. Se a entrada for True, a saída será False e, quando a entrada for False, a saída será True.

Você já ouviu falar da Tabela Verdade? Ela é uma ferramenta utilizada para verificarmos a validade lógica de uma proposição. Nesta tabela, temos colunas com as possíveis entradas de dados e a saída.

A	not A
True	False
False	True

Tabela 1 – Tabela verdade operador not (não)
Fonte: Elaborado pelo autor (2021)

O operador **and** (e) também é conhecido como conjunção e terá como saída o valor True apenas se ambos os operandos forem True. Nos demais casos de combinações de entradas, a saída será False.

A	B	A and B
False	False	False
False	True	False
True	False	False
True	True	True

Tabela 2 – Tabela verdade operador and (e)
Fonte: Elaborado pelo autor (2021)

Outro operador lógico que podemos utilizar na linguagem é o operador **or** (ou), também conhecido como disjunção. Ele possui a seguinte regra: a saída será False apenas se ambos os operandos forem False. Nos demais casos, a saída será True.

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

Tabela 3 – Tabela verdade operador or (ou)
Fonte: Elaborado pelo autor (2021)

2.5 Operadores de associação

Os operadores de associação são utilizados para averiguar se um objeto está contido em uma sequência. Por exemplo, podemos verificar se uma letra ou palavra está contida em um texto ou ainda se um número está em uma lista.

Operador	Exemplo
in	"s" in resposta
not in	"s" not in resposta

Quadro 5 – Operadores de associação
Fonte: Elaborado pelo autor (2021)

```
resposta = "sim"
texto = "Brasil"
print("s" in resposta)
print("n" not in resposta)
print("b" in texto)
print("B" in texto)
```

Código-fonte 11 – Utilização de operadores de associação
Fonte: Elaborado pelo autor (2021)

2.6 Operadores de identidade

Quando precisamos comparar se dois objetos utilizam a mesma posição de memória no Python, usamos os operadores de identidade: `is` e `not is`. O operador de identidade é diferente do operador relacional de igualdade `==`, pois o operador de identidade analisa o endereço e o de igualdade analisa o valor.

Operador	Exemplo
is	n1 is n2
is not	n1 is not n2

Quadro 6 – Operadores de identidade
Fonte: Elaborado pelo autor (2021)

O método `id()` retorna o endereço de memória de um objeto, podemos utilizá-lo em conjunto com o comando `print` para exibir este endereço em tela.

```
cidade_p1 = "São Paulo"
cidade_p2 = "São Paulo"
cidade_p3 = "Rio de Janeiro"
print(id(cidade_p1))
print(id(cidade_p2))
print(id(cidade_p3))
print(cidade_p1 is cidade_p2)
```

```
print(cidade_p1 is not cidade_p3)
print(cidade_p1 is cidade_p3)
```

Código-fonte 12 – Utilização de operadores de identidade
 Fonte: Elaborado pelo autor (2021)

2.7 Precedência dos operadores

Para realizar as operações, a linguagem de programação segue um padrão parecido com o da Matemática para definir qual operação será realizada primeiro. O operador que possui maior prioridade será executado primeiro. Em caso de operadores com a mesma prioridade (precedência), a operação é realizada da esquerda para a direita na expressão.

Operador	Função	Precedência
()	Parênteses	Maior prioridade
**	Exponenciação	
+x, -x	Sinal	
*, /, //, %	Multiplicação, divisão, divisão inteira e módulo	
+, -	Soma e subtração	
<, <=, >, >=	Relacionais	
==, !=	Igual e diferente	
is, is not	Identidade	
in, not in	Associação	
not	Não	
and	E	
or	Ou	
=, +=, -=, *=, /=, %=	Atribuição	Menor prioridade

Quadro 7 – Precedência dos operadores
 Fonte: Elaborado pelo autor (2021)

2.8 Hora de exercitar

Agora que aprendemos alguns comandos básicos da linguagem Python e já temos o nosso ambiente configurado e funcionando, que tal fazermos alguns exercícios e aprendermos a lidar com números?

2.8.1 A calculadora!

Nossa missão é simples: criar a mais incrível calculadora de todos os tempos! Ela deve receber dois valores do usuário e realizar as 4 operações básicas: soma, subtração, divisão e multiplicação.

O algoritmo que realiza essas operações deve se aproximar do fluxograma e pseudocódigo a seguir:

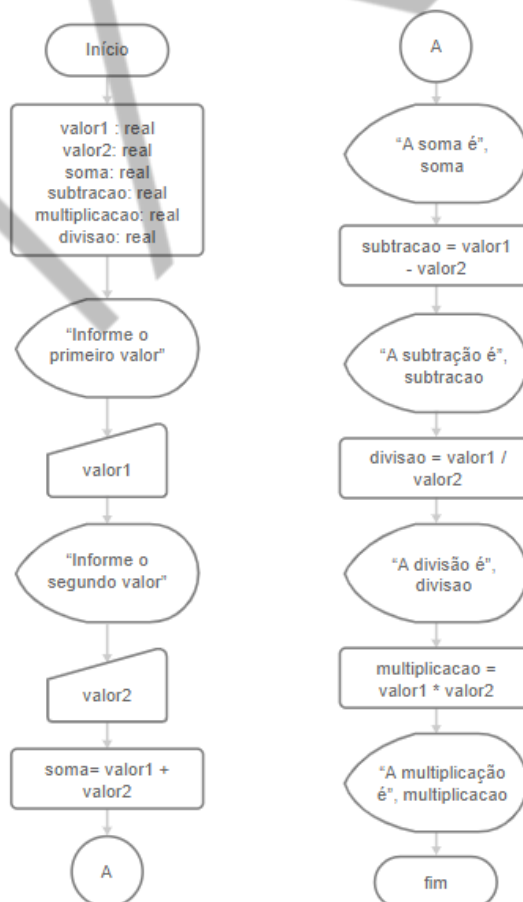


Figura 16 – Fluxograma para o algoritmo de calculadora
Fonte: Elaborado pelo autor (2019)

```
algoritmo "Soma"
variáveis
    valor1,    valor2,    soma,    divisao,    subtracao,
multiplicacao: real
início
    Escreva "Digite o primeiro valor"
    Leia valor1
    Escreva "Digite o segundo valor"
    Leia valor2
    soma = valor1 + valor2
    Escreva "A soma é ", soma
    subtracao = valor1 - valor2
    Escreva "A soma é ", subtracao
    divisao = valor1 / valor2
    Escreva "A soma é ", divisao
    multiplicacao = valor1 * valor2
    Escreva "A soma é ", multiplicacao
Fim
```

Código-fonte 13 – Pseudocódigo para o algoritmo de calculadora

Fonte: Elaborado pelo autor (2019)

Como aprendizado e corrida são coisas diferentes, vamos tentar resolver esse problema por partes? No PyCharm, em um script chamado *calculadora.py*, vamos criar primeiro a parte do código que fará a soma entre dois valores:

```
valor1 = input("Por favor, digite o primeiro valor: ")
valor2 = input("Por favor, digite o segundo valor: ")
soma = valor1 + valor2
print("A soma entre os valores é " + soma)
```

Código-fonte 14 – Código em Python para implementar parte do algoritmo de calculadora

Fonte: Elaborado pelo autor (2019)

Executar o código pode ser um pouco decepcionante! Afinal de contas, veja o resultado:

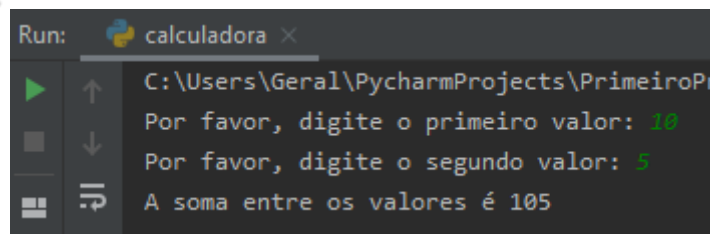


Figura 17 – Execução da soma em Python

Fonte: Elaborado pelo autor (2019)

Por alguma razão, o Python parece considerar as variáveis como *texto* e não como número. Para validar essa suspeita, podemos tentar *printar* na tela o tipo das variáveis *valor1* e *valor2*. Isso pode ser feito com o comando *type*. Veja:

```
valor1 = input("Por favor, digite o primeiro valor: ")
valor2 = input("Por favor, digite o segundo valor: ")
soma = valor1 + valor2
print("A soma entre os valores é " + soma)
print(type(valor1))
print(type(valor2))
```

Código-fonte 15 – Uso do comando *type* para verificar o tipo das variáveis
Fonte: Elaborado pelo autor (2019)

A execução desse script nos mostra que as variáveis *valor1* e *valor2* estão sendo tratadas como *strings* (str), ou seja, como texto.

Para que elas passem a ser tratadas como números, temos que convertê-las, podendo usar os tipos *int* (números inteiros), *float* (números com casas decimais) e *complex* (números complexos).

Tipos Básicos	Descrição	Exemplo
int	Números inteiros	1, 2, 100
float	Números reais (ponto flutuante)	1.5, 2.07, 50.29
complex	Números complexos	4j, 5+2j, 15j
bool	Valores lógicos	True, False, 1, 0
string	Textos	"Texto", "a", "10"

Quadro 8 – Tipos de dados básicos
Fonte: Elaborado pelo autor (2021)

Para fazer a conversão, devemos escrever o tipo de dado e colocar a variável entre parênteses. Portanto, para que as variáveis *valor1* e *valor2* sejam variáveis que aceitam números decimais, no momento da soma, podemos fazer a conversão.

Ao fazer isso, porém, a variável *soma* passará a ser do tipo *float*, então, temos duas alternativas: precisamos converter essa variável para texto na hora de exibi-la novamente ou usar um recurso do Python que nos permite escrever um texto, indicando os locais onde incluiremos valores de variáveis.

A segunda alternativa utiliza melhor a linguagem Python e evita que o programador faça uma série de conversões manualmente. No nosso caso, para exibir a mensagem: “A soma entre os valores é” e incluir a variável *soma*, indicaremos o texto da seguinte forma: “A soma entre os valores é {}” e usaremos a função *format* para indicar que as chaves devem ser substituídas por uma variável.

O código final da soma será:

```
valor1 = input("Por favor, digite o primeiro valor: ")
valor2 = input("Por favor, digite o segundo valor: ")
soma = float(valor1) + float(valor2)
print("A soma entre os valores é {}".format(soma))
```

Código-fonte 16 – Conversão de tipos de dados em Python

Fonte: Elaborado pelo autor (2019)

Se implementarmos as demais partes do nosso algoritmo, teremos:

```
valor1 = input("Por favor, digite o primeiro valor: ")
valor2 = input("Por favor, digite o segundo valor: ")
soma = float(valor1) + float(valor2)
print("A soma entre os valores é {}".format(soma))
subtracao = float(valor1) - float(valor2)
print("A subtração entre os valores é {}".format(subtracao))
divisao = float(valor1) / float(valor2)
print("A divisão entre os valores é {}".format(divisao))
multiplicacao = float(valor1) * float(valor2)
print("A multiplicação entre os valores é {}".format(multiplicacao))
```

Código-fonte 17 – Calculadora totalmente implementada em Python

Fonte: Elaborado pelo autor (2019)

3 O PATINETE ELÉTRICO!

A cidade de São Paulo conta com centenas de patinetes elétricos espalhados em pontos estratégicos. Vez por outra, surge algum incidente leve ou acidente grave que faz as pessoas questionarem a velocidade desses veículos.

Mais do que a velocidade máxima, nos interessa a velocidade média! É por isso que criaremos um algoritmo capaz de calcular a velocidade média a partir de duas informações: a distância que o patinete percorreu e o tempo que demorou para isso.

Lembrando das aulas de Física que tivemos na escola, é possível compreender que o algoritmo consiste em armazenar a distância e o tempo em variáveis numéricas e realizar a divisão, exibindo o cálculo ao final. Portanto, faremos a implementação em Python:

```
print("Esse programa calcula a velocidade média de um patinete")
distancia = input("Qual foi a distância em metros percorrida pelo patinete? ")
tempo = input("Quantos minutos o patinete demorou para percorrer essa distância? ")
velocidade_media = float(distancia) / float(tempo)
print("O patinete atingiu uma velocidade de {} m/min".format(velocidade_media))
```

Código-fonte 18 – Calculadora totalmente implementada em Python
Fonte: Elaborado pelo autor (2019)

O algoritmo está correto e o programa foi bem implementado, mas um bom programador busca sempre o melhor. Neste programa, o usuário pode digitar valores que gerem um resultado com muitos dígitos decimais. Podemos limitar essa exibição de dígitos com um pequeno truque de formatação!

Dentro das chaves que apontam onde a variável *velocidade_media* será exibida, podemos indicar quantas casas devem ser mostradas após a vírgula. Para exibir 2 casas, indicaremos {0:.2f}, para exibir 1 casa {0:.1f} e assim por diante.

Vejamos o código final:

```
print("Esse programa calcula a velocidade média de um patinete")
distancia = input("Qual foi a distância em metros
```

```
percorrida pelo patinete? ")
    tempo = input("Quantos minutos o patinete demorou para
percorrer essa distância? ")
    velocidade_media = float(distancia) / float(tempo)
    print("O patinete atingiu uma velocidade de {0:.2f}
m/min".format(velocidade_media))
```

Código-fonte 19 – Calculadora totalmente implementada
em Python com arredondamento de valores
Fonte: Elaborado pelo autor (2019)

3.1 Vamos trocar?

Uma das partes mais legais de começar a programar é resolver pequenos problemas algorítmicos que servem para exercitar a lógica, muito mais do que para resolver um problema real.

Veja esse caso: imagine um caso em que o usuário digite um valor na variável A e outro valor na variável B, e o algoritmo deve inverter os conteúdos das variáveis!

Como fazer isso sem que, logo na primeira troca, o valor de uma das variáveis não seja perdido?

Para resolver esse problema, como tantos em programação, existem inúmeras alternativas. A mais simples delas é a criação de uma terceira variável temporária. A ideia é simples: assim como quando estamos com um prato de comida em cada mão precisamos de uma terceira pessoa para nos ajudar na inversão, precisaremos de uma terceira variável aqui.

Logo teremos:

```
print("Esse programa inverte os conteúdos de duas
variáveis")
A = input("Digite o conteúdo da variável 1: ")
B = input("Digite o conteúdo da variável 2: ")
troca = A
A = B
B = troca
print("Agora que trocamos, a variável A contém {} e a
variável B contém {}".format(A, B))
```

Código-fonte 20 – Calculadora totalmente implementada
em Python com arredondamento de valores
Fonte: Elaborado pelo autor (2019)

Chegamos ao final da primeira fase conhecendo como um software é criado, quais técnicas utilizar em seu desenvolvimento e começamos a conhecer um pouco de programação.

Prepare-se! Na próxima fase, teremos mais programação e design. Até lá!

EMSE

REFERÊNCIAS

PIVA, D. J. **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, S.; RISSETTI, G. **Lógica de programação e estrutura de dados**. São Paulo: Pearson Prentice Hall, 2009.

RAMALHO, L. **Python fluente**: programação clara, concisa e eficaz. São Paulo: Novatec, 2015.

EMEND