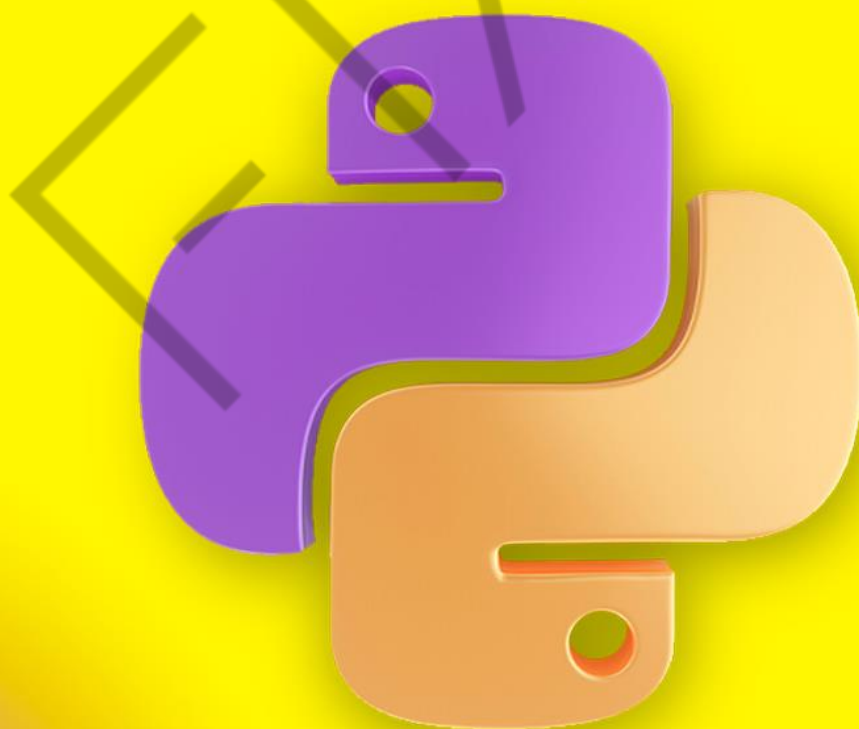


PYTHON DEVELOPMENT

VARIAVEIS E TIPOS DE DADOS



2

LISTA DE FIGURAS

Figura 1 – Criação de arquivos de texto no Windows ou Linux/macOS	7
Figura 2 – Execução do script via linha de comando no Windows ou Linux/macOS	7
Figura 3 – Mensagem de compartilhamento de dados no PyCharm.....	8
Figura 4 – Use o botão New Project para criar um novo projeto	8
Figura 5 – O nome do projeto é o último texto após a barra na caixa Location	9
Figura 6 – Em sistemas com mais de um interpretador instalado é necessário selecionar o correto.....	10
Figura 7 – Conferência de todos os itens para a criação do novo projeto.....	11
Figura 8 – Criação de um novo script.....	11
Figura 9 – Código escrito dentro do PyCharm	12
Figura 10 – Opção Run para executar o script.....	12
Figura 11 – Programa executando no terminal do PyCharm.....	13
Figura 12 – Criação do script calculadora_inteiros.py.....	14
Figura 13 – Resultado incorreto da soma	19
Figura 14 – Exibição do tipo de dados da variável primeiro_valor	17
Figura 15 – Erro ao concatenar int e str.....	18
Figura 16 – Soma funcionando após utilizar os tipos corretos	19
Figura 17 – Programa executando com resultado com diversas casas decimais	22
Figura 18 – Programa proposto sendo executado	27

LISTA DE QUADROS

Quadro 1 – Operadores aritméticos em Python	19
Quadro 2 – Operadores aritméticos em Python	24

EMENDAS

LISTA DE CÓDIGOS-FONTE

Código-fonte 1 – Programa de boas vindas em Python	6
Código-fonte 2 – Início do script calculadora_inteiros.py	18
Código-fonte 3 – Primeira tentativa de soma no script calculadora_inteiros.py	19
Código-fonte 4 – Segunda tentativa de soma no script calculadora_inteiros.py	17
Código-fonte 5 – Primeira tentativa de soma no script calculadora_inteiros.py	18
Código-fonte 6 – Versão final do programa calculadora_inteiros.py	20
Código-fonte 7 – Versão inicial do programa velocidade_media.py	21
Código-fonte 8 – Versão inicial do programa velocidade_media.py	22
Código-fonte 9 – Criando e exibindo variáveis bool	23
Código-fonte 10 – Testando todos os operadores bool	24
Código-fonte 11 – Criando uma variável de texto com quebra de linha e tabulação	25
Código-fonte 12 – Testando métodos da str	29
Código-fonte 13 – Unindo o que estudamos até agora	28

SUMÁRIO

VARIÁVEIS E TIPOS DE DADOS.....	6
1 O QUE VOCÊ ESTÁ AGUARDANDO? PARTINDO PARA A IDE	6
2 VÁRIAS VARIÁVEIS (E TIPOS DE DADOS)	13
2.1 Um dado inteiro para você	14
2.2 Flutuando por aí	20
2.3 Bool? É lógico!	23
2.4 Eu testo o texto	25
3 VAMOS JUNTAR TUDO?	27
REFERÊNCIAS.....	29

VARIÁVEIS E TIPOS DE DADOS

1 O QUE VOCÊ ESTÁ AGUARDANDO? PARTINDO PARA A IDE

Se você chegou até aqui, certamente não está perdido no mundo do Python e já pode se considerar um estudante desta linguagem: você já sabe quais são os principais poderes e usos do Python, tem um ambiente de desenvolvimento totalmente configurado e já sabe até mesmo criar programas que façam leitura e exibição de dados de texto. Não há mais nada a aprender, certo?

Claro que ainda há muito a aprender e vamos começar esta etapa encarando um problema sutil que você pode já ter detectado: o quão prático é ficar escrevendo os seus programas no terminal? A resposta é que isso é tão prático quanto tentar cozinhar com os clientes do seu restaurante tentando pegar a comida diretamente do fogão.

Para contornar este problema, é muito mais eficaz escrevermos todo o nosso código em um arquivo e, quando ele estiver finalizado, solicitarmos ao Python que interprete e execute este arquivo. Ele será o nosso *script*.

Vamos partir do código abaixo:

```
nome = input("Por favor, informe o seu nome: ")  
print("Bom dia " + nome)
```

Código-fonte 1 – Programa de boas-vindas em Python
Fonte: Elaborado pelo autor (2022)

Poderíamos salvar este código-fonte em um arquivo de texto, utilizando qualquer editor de nossa preferência, com uma extensão `.py` e, depois, utilizar o terminal para abrir a mesma pasta onde o arquivo se encontra e executar a instrução `python boas_vindas.py` no Windows ou `python3 boas_vindas.py` no Linux ou macOS.

Varia?veis e tipos de dados

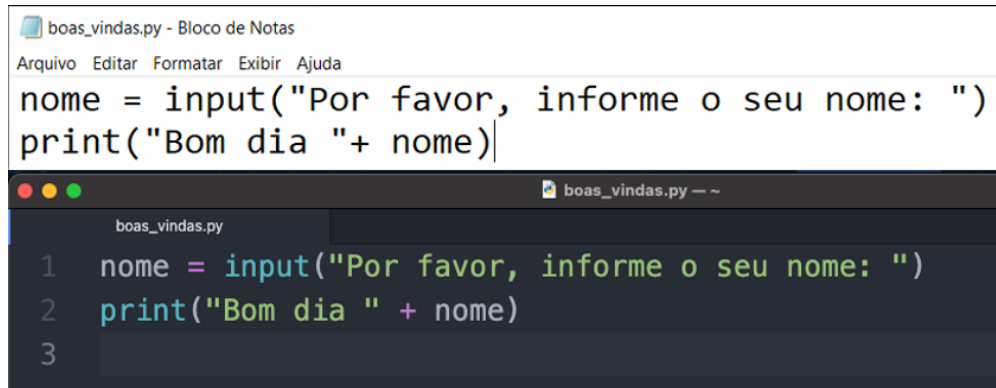


Figura 1 – Criação de arquivos de texto no Windows ou Linux/macOS
Fonte: Elaborado pelo autor (2022)

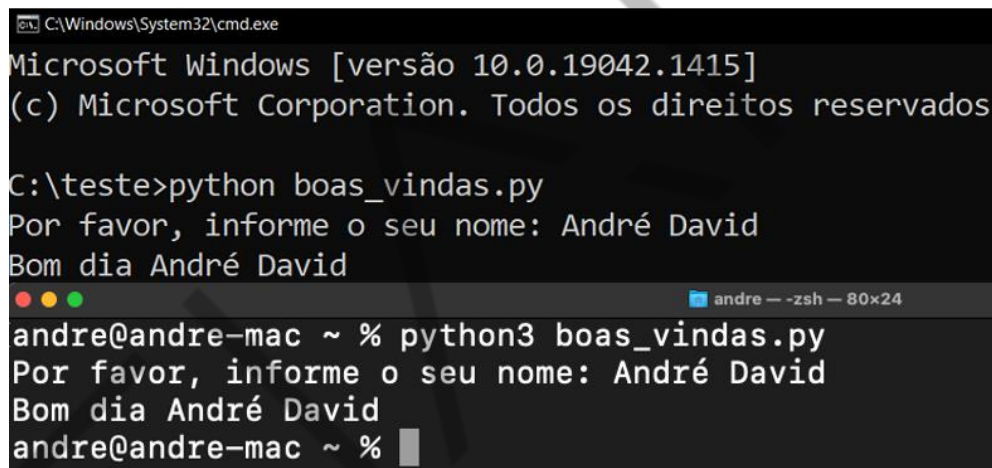


Figura 2 – Execução do script via linha de comando no Windows ou Linux/macOS
Fonte: Elaborado pelo autor (2022)

Apesar de esta solução funcionar e ser muito útil em cenários onde não se tem acesso a uma IDE, utilizar um programa específico para desenvolvimento traz muitas vantagens. Por essa razão, vamos abrir o PyCharm.

Na primeira execução do PyCharm podemos ser perguntados se desejamos compartilhar dados com a JetBrains para a melhoria do produto, como na figura abaixo. Caso queira compartilhar dados, selecione a opção “Send Anonymous Statistics”. Caso não deseje enviar dados, selecione a opção “Don’t Send”.

Varia?veis e tipos de dados

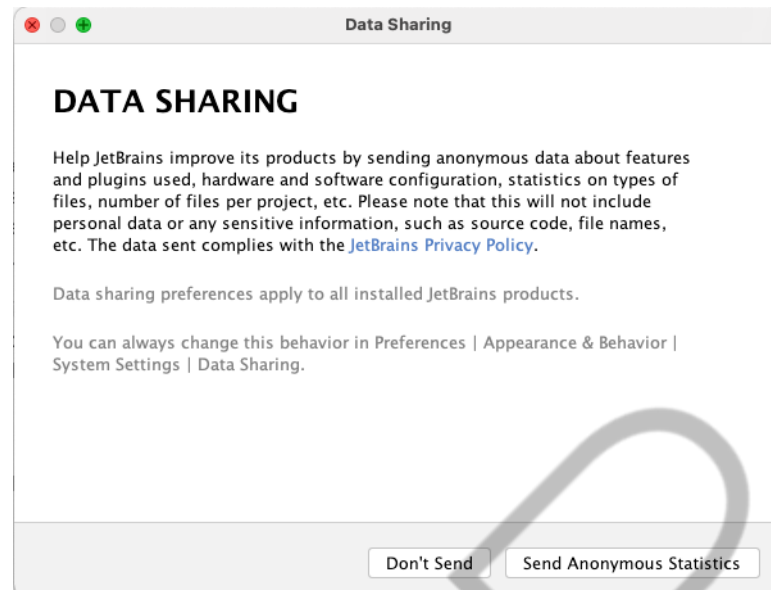


Figura 3 – Mensagem de compartilhamento de dados no PyCharm
Fonte: Elaborado pelo autor (2022)

Uma vez que o PyCharm for aberto, clique na opção “New Project” para criar um novo projeto.

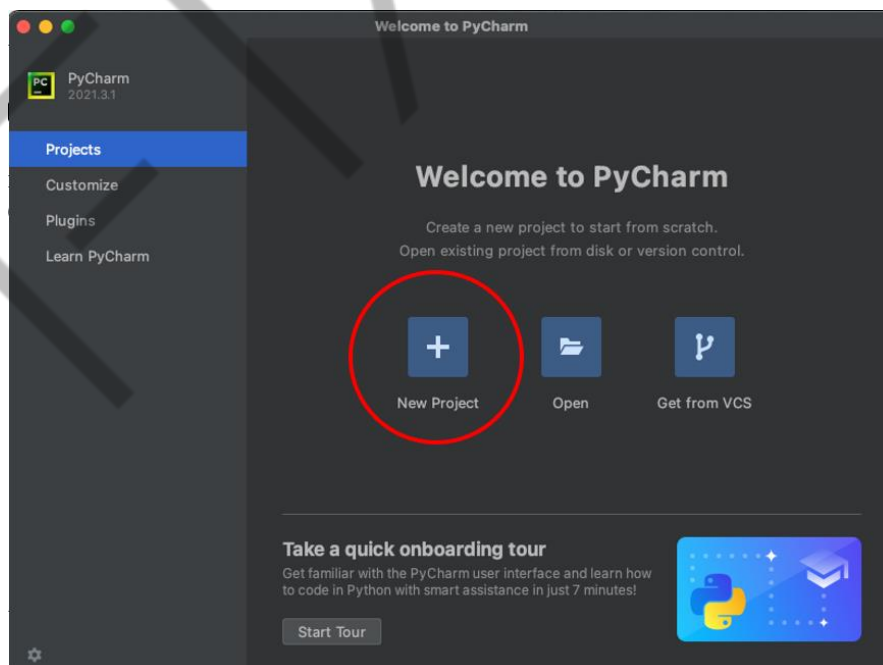


Figura 4 – Use o botão New Project para criar um novo projeto
Fonte: Elaborado pelo autor (2022)

Varia?veis e tipos de dados

Tanto no Windows quanto no Linux e no macOS, devemos utilizar a caixa “Location” para definir o local onde a pasta do projeto será criada. O último nome após a última barra é o nome do seu projeto. Chamaremos este projeto de entendendoVariaveis.

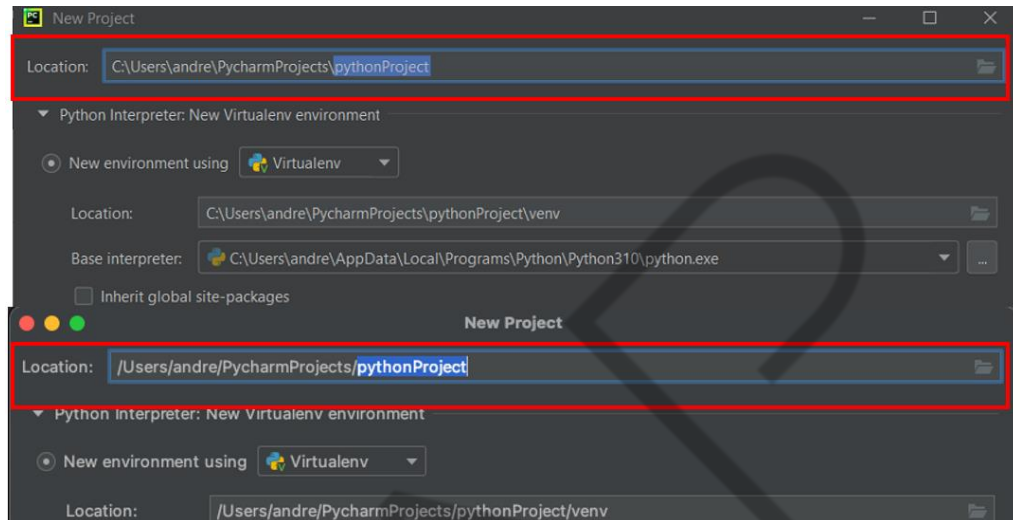


Figura 5 – O nome do projeto é o último texto após a barra na caixa Location
Fonte: Elaborado pelo autor (2022)

O PyCharm possui como recurso a criação automática de um ambiente virtual. Isso quer dizer que os pacotes que forem instalados neste projeto serão instalados apenas nele, sem afetar o seu computador e os demais projetos que você crie. É importante ficar atento apenas ao interpretador que será utilizado: caso você possua mais de um instalado (como em distribuições Linux ou no macOS, que possui a versão 2.x e 3.x instalada), é necessário escolher a versão correta. A figura abaixo demonstra a escolha da versão adequada do interpretador do Python em um computador com macOS.

Variações e tipos de dados

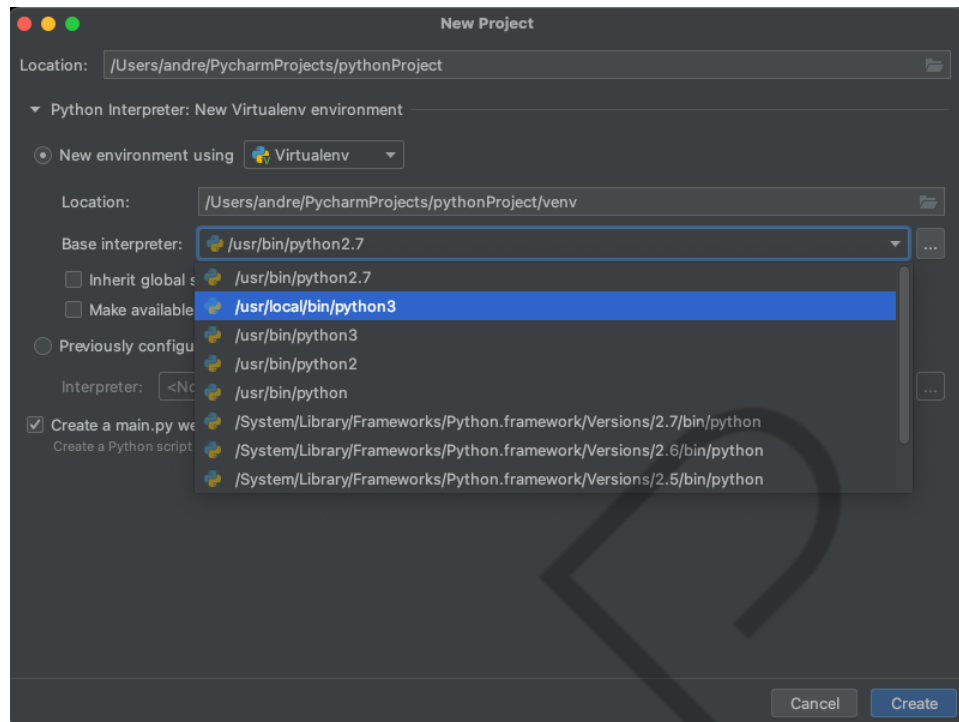


Figura 6 – Em sistemas com mais de um interpretador instalado é necessário selecionar o correto
Fonte: Elaborado pelo autor (2022)

Por fim, devemos desmarcar a caixa que contém o texto “Create a main.py welcome script”, porque não queremos que o PyCharm crie um arquivo de exemplo. Tanto no macOS quanto no Linux e no Windows devemos ter uma tela parecida com a imagem abaixo (guardadas as particularidades de representação de pastas de cada sistema).

Varia?veis e tipos de dados

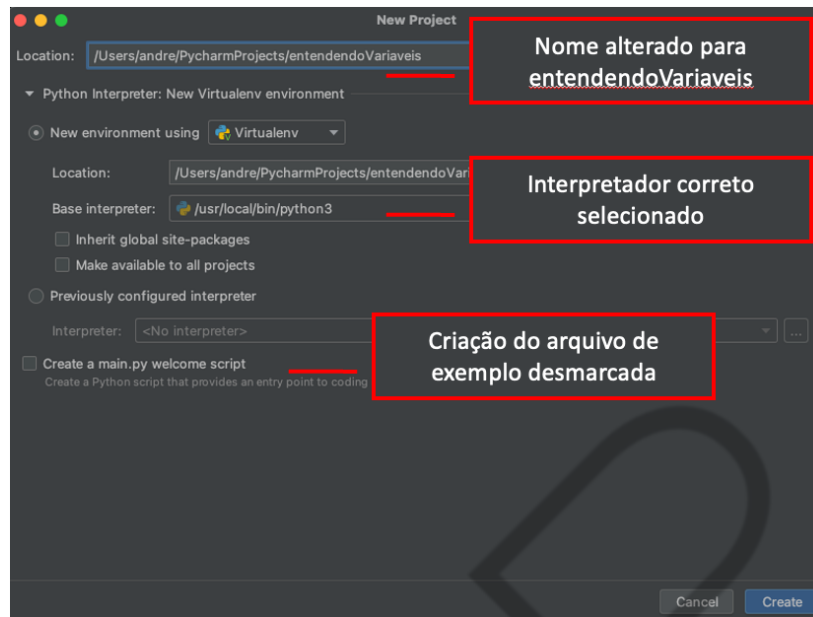


Figura 7 – Conferência de todos os itens para a criação do novo projeto
Fonte: Elaborado pelo autor (2022)

Basta apenas clicarmos no botão “Create” e o projeto será criado!

Quando o PyCharm for carregado, teremos todas as pastas do nosso projeto do lado esquerdo da tela. Nossa primeira tarefa será criar o script que conterá nosso código. Para isso basta clicar no nome do projeto *entendendoVariaveis* com o botão direito do mouse, escolher a opção *New* → *Python File*, dando ao arquivo o nome de *boas_vindas.py*

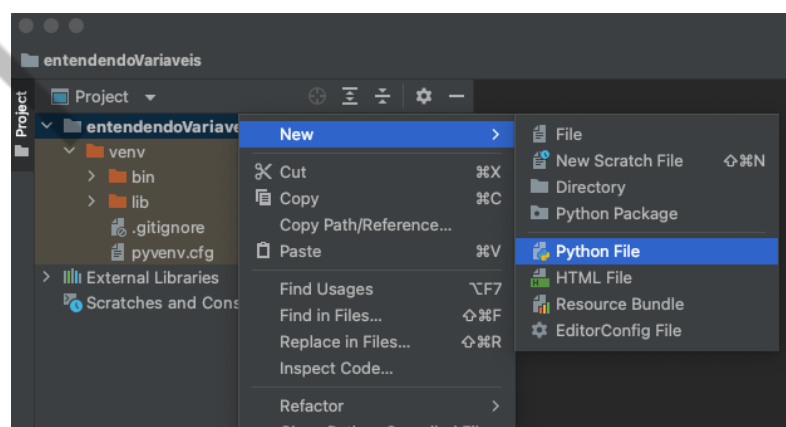


Figura 8 – Criação de um novo script
Fonte: Elaborado pelo autor (2022)

Variações e tipos de dados

Vamos agora inserir o mesmo código desenvolvido anteriormente, que recebe o nome do usuário e apresenta uma mensagem de boas-vindas:

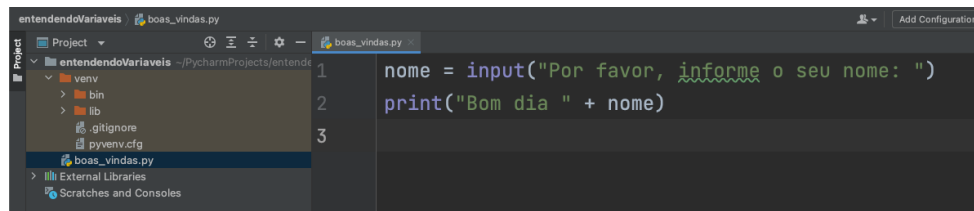


Figura 9 – Código escrito dentro do PyCharm
Fonte: Elaborado pelo autor (2022)

Logo de cara podemos notar uma vantagem de usar uma IDE: nosso código ficou formatado com cores que facilitam a compreensão do que são variáveis, o que são funções e o que são textos.

Para executarmos o código de uma só vez, basta clicarmos no nome do arquivo com o botão direito do mouse e selecionarmos a opção *Run*, conforme a imagem a seguir:

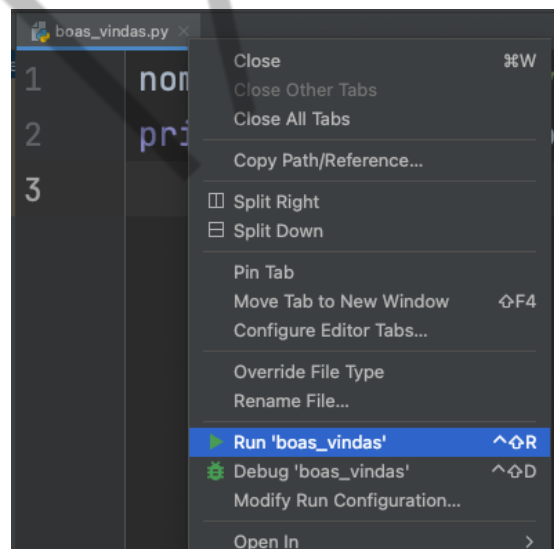


Figura 10 – Opção Run para executar o script
Fonte: Elaborado pelo autor (2022)

Varia?veis e tipos de dados

Essa opção fará com que nosso script seja executado em um pequeno terminal na parte inferior do PyCharm:

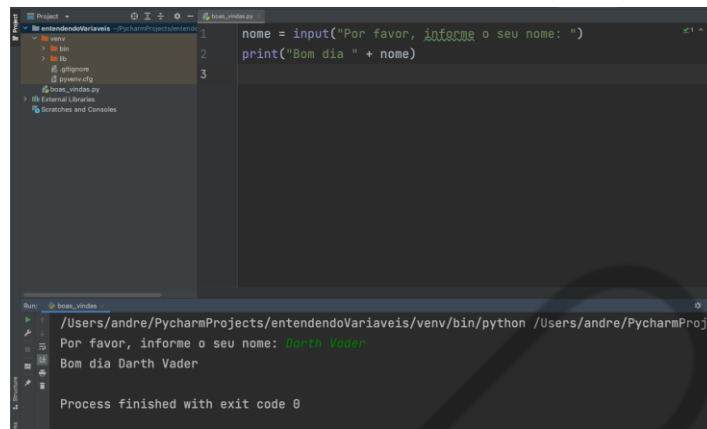


Figura 11 – Programa executando no terminal do PyCharm
Fonte: Elaborado pelo autor (2022)

Agora que você já sabe que é possível escrever um programa completo e depois executá-lo, e também como criar esses scripts dentro do PyCharm, que tal avançarmos um pouco mais no mundo da programação?

2 VÁRIAS VARIÁVEIS (E TIPOS DE DADOS)

Responda rapidamente: qual é a semelhança entre uma fotografia do Faustão e o título de um livro registrado no sistema da Amazon? Exatamente o que você pensou: nenhuma!

Se quisermos ser muito detalhistas, podemos dizer que tanto a fotografia quanto o título do livro são números binários, organizados de uma forma que o computador consiga representar aquilo que vemos na tela. Na prática, porém, é essa organização diferente dos dados que nos importa.

Para cada tipo de dado que um software pode representar, temos também um tipo de variável diferente, e a razão é muito fácil de compreender: imagine que você precise armazenar o nome de um usuário e seu ano de nascimento. Você pode fazer operações matemáticas com o ano de nascimento (como utilizar para calcular a idade) que não pode fazer com o nome, certo? Para que o Python consiga fazer as operações corretas, estes dados têm tipos diferentes.

Podemos dividir os tipos de dados de muitas formas diferentes, mas para facilitar vamos considerar dados numéricos, lógicos e de texto.

Vamos começar com os dados numéricos!

2.1 Um dado inteiro para você

Os dados numéricos servem para representar números em um computador e, se as aulas de matemática não deixaram grandes traumas, você deve se recordar que os números podem ser de vários conjuntos numéricos: inteiros, fracionários, irracionais, imaginários e.... ei! Não precisa pular este capítulo. Vamos devagar.

Por mais que você não queira nem imaginar um número imaginário, é fácil entender o que é um número inteiro: é aquele que não tem parte decimal, que não tem vírgula. O ano do seu nascimento é um número inteiro, sua idade também, assim como a quantidade de filhos que um casal tem.

Para entender bem o que implica trabalhar com números inteiros, vamos imaginar um cenário simples: precisamos criar uma calculadora que receba dois números inteiros digitados pelo usuário e realize as 4 operações básicas entre eles.

Para começar este desafio, vamos criar um script chamado *calculadora_inteiros.py*. Para fazer isso basta clicar com o botão direito no nome do projeto, selecionar a opção *New* → *Python File*, como fizemos anteriormente.

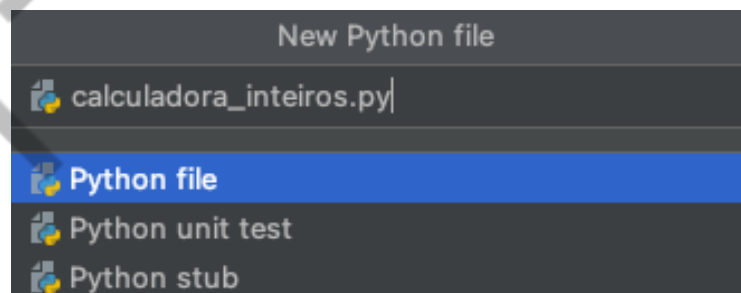


Figura 12 – Criação do script *calculadora_inteiros.py*
Fonte: Elaborado pelo autor (2022)

Variações e tipos de dados

O programa que vamos escrever será um algoritmo, ou seja, uma sequência de passos lógicos e estruturados que atingem um determinado objetivo. Para o nosso algoritmo, podemos desenhar a seguinte sequência:

1. Dar boas-vindas ao usuário
2. Solicitar que o usuário digite o primeiro número
3. Solicitar que o usuário digite o segundo número
4. Realizar o cálculo da soma
5. Exibir o resultado da soma
6. Realizar o cálculo da subtração
7. Exibir o resultado da subtração
8. Realizar o cálculo da multiplicação
9. Exibir o resultado da multiplicação
10. Realizar o cálculo da divisão
11. Exibir o resultado da divisão

Com o que você aprendeu até agora, é capaz de escrever o script que realiza os 3 primeiros passos sem problemas. Seu código deve ficar parecido com este:

```
print("Bem vindo à calculadora de inteiros!")
primeiro_valor = input("Por favor, insira o primeiro valor: ")
segundo_valor = input("Por favor, insira o segundo valor: ")
```

Código-fonte 2 – Início do script calculadora_inteiros.py
Fonte: Elaborado pelo autor (2022)

Para realizarmos a soma entre os valores, você deve utilizar o símbolo de soma que já está acostumado, o +. Se continuarmos nosso script, portanto, devemos ter algo próximo ao seguinte:

Variações e tipos de dados

```
print("Bem vindo à calculadora de inteiros!")
primeiro_valor = input("Por favor, insira o primeiro
valor: ")
segundo_valor = input("Por favor, insira o segundo valor:
")

soma = primeiro_valor + segundo_valor
print("A soma entre os valores é " + soma)
```

Código-fonte 3 – Primeira tentativa de soma no script calculadora_inteiros.py

Fonte: Elaborado pelo autor (2022)

Nosso script realiza uma operação entre as variáveis `primeiro_valor` e `segundo_valor` e armazena o resultado na variável `soma`, que depois é exibida em um `print`. Mas será que este código nos dá o resultado esperado? Vejamos:

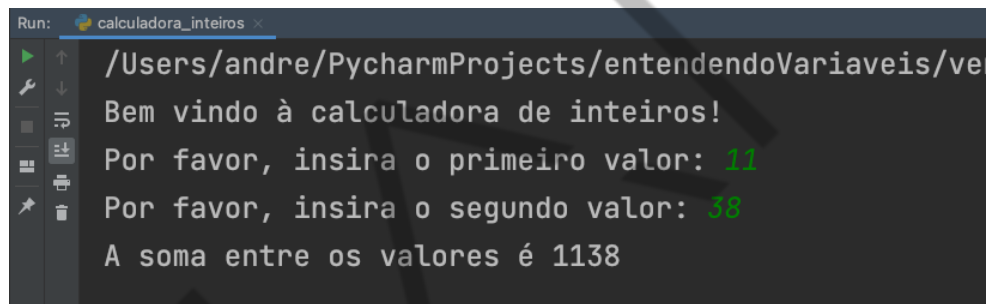


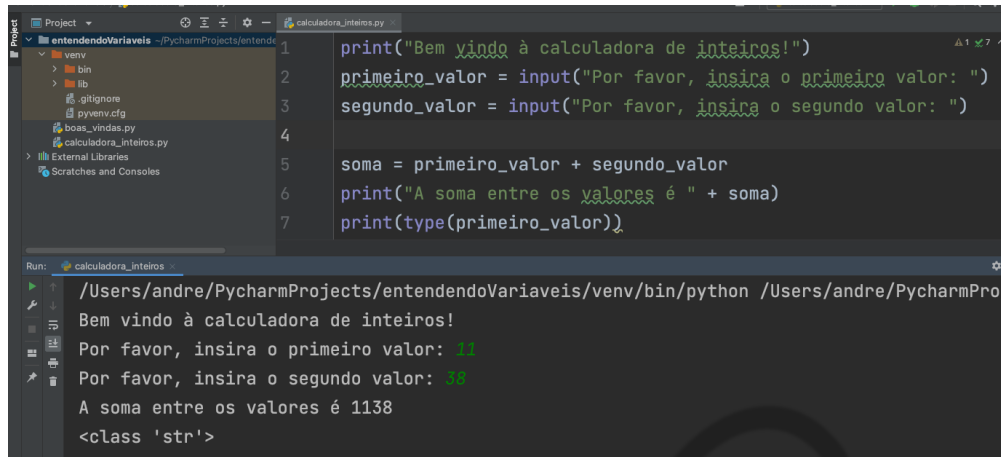
Figura 13 – Resultado incorreto da soma

Fonte: Elaborado pelo autor (2022)

Opa, alguma coisa está errada! Afinal, digitando 11 e 38, nosso resultado deveria ser 49, mas o terminal está exibindo 1138. Aparentemente, o comportamento que foi adotado pelo operador `+` foi de “juntar” o primeiro número ao segundo, ao invés de somá-los. E até agora só havíamos presenciado este comportamento quando “juntamos” um texto a uma variável. E é exatamente isso que está acontecendo!

O Python não está interpretando o conteúdo das variáveis `primeiro_valor` e `segundo_valor` como numérico, e sim como texto. E podemos comprovar isso ao executarmos um `print` que exiba o tipo de dados da variável, acrescentando temporariamente a instrução `print(type(primeiro_valor))` na última linha do nosso código e executando o programa novamente:

Variações e tipos de dados



The screenshot shows a PyCharm IDE with a project named 'entendendoVariaveis'. The file 'calculadora_inteiros.py' is open, showing the following code:

```
1 print("Bem vindo à calculadora de inteiros!")
2 primeiro_valor = input("Por favor, insira o primeiro valor: ")
3 segundo_valor = input("Por favor, insira o segundo valor: ")
4
5 soma = primeiro_valor + segundo_valor
6 print("A soma entre os valores é " + soma)
7 print(type(primeiro_valor))
```

The Run console at the bottom shows the execution output:

```
/Users/andre/PycharmProjects/entendendoVariaveis/venv/bin/python /Users/andre/PycharmPro
Bem vindo à calculadora de inteiros!
Por favor, insira o primeiro valor: 11
Por favor, insira o segundo valor: 10
A soma entre os valores é 1138
<class 'str'>
```

Figura 14 – Exibição do tipo de dados da variável `primeiro_valor`
Fonte: Elaborado pelo autor (2022)

A função `type` nos revelou que a variável `primeiro_valor` é do tipo `str` (texto, como veremos mais adiante), e não do tipo inteiro. O tipo inteiro em Python é o tipo `int`.

A confusão de tipos ocorreu porque a função `input()` considera que qualquer valor fornecido é do tipo texto. Para corrigirmos este problema, faremos uma *conversão*, envolvendo toda a função de `input` entre parênteses e escrevendo `int` antes dela. O código abaixo já possui a leitura corrigida e, temporariamente, possui a exibição do tipo de dados que selecionamos.

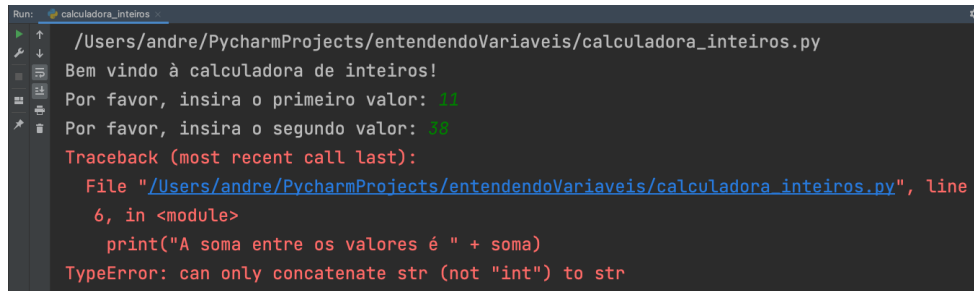
```
print("Bem vindo à calculadora de inteiros!")
primeiro_valor = int(input("Por favor, insira o primeiro
valor: "))
segundo_valor = int(input("Por favor, insira o segundo
valor: "))

soma = primeiro_valor + segundo_valor
print("A soma entre os valores é " + soma)
print(type(primeiro_valor))
```

Código-fonte 4 – Segunda tentativa de soma no script `calculadora_inteiros.py`
Fonte: Elaborado pelo autor (2022)

Agora é só mandar o programa executar e sua vida será só de alegrias:

Varia?veis e tipos de dados



```
Run: calculadora_inteiros
/Users/andre/PycharmProjects/entendendoVariaveis/calculadora_inteiros.py
Bem vindo à calculadora de inteiros!
Por favor, insira o primeiro valor: 11
Por favor, insira o segundo valor: 38
Traceback (most recent call last):
  File "/Users/andre/PycharmProjects/entendendoVariaveis/calculadora_inteiros.py", line
    6, in <module>
      print("A soma entre os valores é " + soma)
TypeError: can only concatenate str (not "int") to str
```

Figura 15 – Erro ao concatenar int e str
Fonte: Elaborado pelo autor (2022)

Parece que não tivemos só alegrias, não é mesmo? O interpretador retornou um erro que indica que só pode concatenar (unir, “juntar”) dois textos (str), e não um texto (str) com um inteiro (int).

Esse erro está acontecendo pela má prática de utilizar o operador de + para unir um texto a uma variável: se essa variável também for de texto, como no código presente no início deste capítulo, não veremos nenhum erro. No entanto, se a variável não for de texto, você terá um belo erro em mãos.

Mas não se preocupe! Após prometer que nunca mais utilizará o operador de + para concatenar textos e variáveis, vamos utilizar uma forma mais adequada: o f-string.

Para utilizar esta notação, basta colocar uma letra *f* antes do seu texto iniciar e, dentro das aspas, quando for necessário exibir uma variável faremos isso entre chaves (`{}`). Por exemplo: `print(f“O seu nome é {nome}”)`. Veja como fica o nosso código adequado a essa prática:

```
print("Bem vindo à calculadora de inteiros!")
primeiro_valor = int(input("Por favor, insira o primeiro
valor: "))
segundo_valor = int(input("Por favor, insira o segundo
valor: "))

soma = primeiro_valor + segundo_valor
print(f"A soma entre os valores é {soma}")
```

Código-fonte 5 – Primeira tentativa de soma no script calculadora_inteiros.py
Fonte: Elaborado pelo autor (2022)

```
/Users/andre/PycharmProjects/entendendo  
 Bem vindo à calculadora de inteiros!  
 Por favor, insira o primeiro valor: 11  
 Por favor, insira o segundo valor: 38  
 A soma entre os valores é 49
```

Figura 16 – Soma funcionando após utilizar os tipos corretos
Fonte: Elaborado pelo autor (2022)

Sucesso! Com as alterações feitas até aqui, nosso programa funciona muito bem e nos retorna os resultados adequados. Muito obrigado f-string pelos serviços prestados!

DICA: Além do f-string, você pode utilizar o método format, que foi introduzido no python 2.6. Para isso basta escrever um texto e indicar os locais onde devem aparecer variáveis entre chaves. Ao final do texto, escrevemos `.format(nome_da_variável)`. Assim:

```
print("A soma entre os valores é {}".format(soma))
```

Passado o momento de comemoração, precisamos continuar a implementar as demais operações aritméticas na nossa calculadora. O quadro abaixo apresenta os operadores matemáticos que você pode utilizar em Python:

Operador	O que retorna?
+	Soma entre dois valores numéricos
-	Subtração entre dois valores numéricos
*	Multiplicação entre dois valores numéricos
/	Divisão entre dois valores numéricos
%	Resto da divisão entre dois valores numéricos
**	Exponenciação entre dois valores numéricos
//	Parte inteira da divisão entre dois valores numéricos

Quadro 1 – Operadores aritméticos em Python
Fonte: Elaborado pelo autor (2022)

Com base no quadro anterior você pode elaborar o restante do seu programa, encontrando um resultado próximo ao seguinte:

```
print("Bem vindo à calculadora de inteiros!")
primeiro_valor = int(input("Por favor, insira o primeiro
valor: "))
segundo_valor = int(input("Por favor, insira o segundo
valor: "))

soma = primeiro_valor + segundo_valor
print(f"A soma entre os valores é {soma}")

subtracao = primeiro_valor - segundo_valor
print(f"A subtração entre os valores é {soma}")

multiplicacao = primeiro_valor * segundo_valor
print(f"A multiplicação entre os valores é
{multiplicacao}")

divisao = primeiro_valor / segundo_valor
print(f"A divisão entre os valores é {divisao}")
```

Código-fonte 6 – Versão final do programa calculadora_inteiros.py
Fonte: Elaborado pelo autor (2022)

Quem diria que com um punhado de operadores aritméticos e alguns dados do tipo inteiro poderíamos criar o nosso primeiro script que resolve um problema real? Mas não acaba por aí! Afinal, ainda temos mais um tipo numérico para explorar.

2.2 Flutuando por aí

Os dados do tipo *int* podem ser utilizados para solucionar muitos problemas, mas não todos. Afinal, haverá situações em que os seus códigos precisarão receber dados numéricos que possuem casas decimais.

Imagine que você vá escrever o código em Python para o computador de bordo de um novo carro e, para testar o seu código, você resolve escrever um *script* em que a distância e o tempo serão fornecidos pelo usuário, e não pelo sistema do carro.

Esbarramos em um grande problema: a distância percorrida por um carro não será necessariamente um número inteiro, assim como o tempo da viagem também pode conter casas decimais. O tipo numérico que utilizaremos é o *float*.

Variações e tipos de dados

O *float* é o tipo básico utilizado pelo Python para representar números que contenham casas decimais, sejam eles positivos ou negativos. Fora o fato de armazenar números com casas decimais, seu uso é idêntico ao que fizemos do tipo *int*, portanto também faremos a leitura de dados utilizando o `input`, mas nossa conversão será feita com o *float*.

Nosso algoritmo seguirá a seguinte ordem:

1. Dar boas-vindas ao usuário
2. Solicitar a digitação da distância da viagem em quilômetros
3. Solicitar a digitação da distância da viagem em horas
4. Realizar o cálculo da velocidade média (distância dividida pelo tempo)
5. Exibir o resultado obtido

Uma vez que já sabemos exibir uma mensagem (passos 1 e 5), solicitar dados e converter (passos 2 e 3) e realizar uma divisão (passo 4), este script é extremamente simples de escrever. Vamos criar um arquivo chamado `velocidade_media.py` para ele.

```
print("Simulador de computador de bordo")
distancia = float(input("Por favor, informe a distância
percorrida em km "))
tempo = float(input("Por favor, informe o tempo da viagem
em horas "))

velocidade_media = distancia / tempo

print(f"A velocidade média da viagem foi de
{velocidade_media}km/h ")
```

Código-fonte 7 – Versão inicial do programa `velocidade_media.py`
Fonte: Elaborado pelo autor (2022)

O programa anterior funciona muito bem! Ou *quase sempre* funciona muito bem (você já deve ter percebido que a vida de dev tem muitos *quases* envolvidos, certo?): se você digitar valores que resultem em uma dízima periódica, pode acabar com um resultado “feio” na tela:

```
/Users/andre/PycharmProjects/entendendoVariaveis/velocidade.py
Simulador de computador de bordo
Por favor, informe a distância percorrida em km 13
Por favor, informe o tempo da viagem em horas 3
A velocidade média da viagem foi de 4.333333333333333 km/h
```

Figura 17 – Programa executando com resultado com diversas casas decimais
Fonte: Elaborado pelo autor (2022)

Mas podemos resolver esse problema usando os especificadores de formato, presentes na PEP-0498. Para o tipo float, devemos utilizar a sintaxe *valor:dígitos_antes_da_vírgula.dígitos_após_a_vírgula*f. Se quisermos exibir 3 dígitos antes e 3 após a vírgula em uma variável, teremos portanto: `nome_da_variavel:3.3f`. Se quisermos delimitar apenas os dígitos após a vírgula, teremos: `nome_da_variavel:.3f`. O código abaixo limita os dígitos da velocidade média a duas casas decimais:

```
print("Simulador de computador de bordo")
distancia = float(input("Por favor, informe a distância
percorrida em km "))
tempo = float(input("Por favor, informe o tempo da viagem
em horas "))

velocidade_media = distancia / tempo

print(f"A velocidade média da viagem foi de
{velocidade_media:.2f} km/h")
```

Código-fonte 8 – Versão inicial do programa velocidade_media.py
Fonte: Elaborado pelo autor (2022)

Com tudo o que você aprendeu até agora sobre as variáveis numéricas já é possível pensar nos seus próprios programas para solucionar problemas do dia a dia!

Agora é hora de pensarmos no tipo de dado que armazena valores lógicos.

2.3 Bool? É lógico!

Existe um tipo de dados que está por toda parte no nosso computador e representa a própria forma que um computador “pensa”. Estamos falando dos dados do tipo lógico.

Para um computador comum existem apenas dois resultados para uma situação lógica: verdadeiro ou falso. Não há “meio-termo” para este tipo de dados: você está fazendo o curso de Python? Verdadeiro. O autor deste texto tomou café da manhã? Verdadeiro. Kylo Ren é um vilão melhor que Darth Vader? Falso.

Em homenagem a George Boole, matemático e filósofo britânico que criou a álgebra booleana, os tipos lógicos em programação tendem a se chamar boolean ou, no caso do Python, bool.

Enquanto uma variável *int* pode assumir uma série de valores, uma variável *bool* pode conter apenas True (verdadeiro) ou False (falso).

Criando um script chamado `tipo_bool.py` e preenchendo com o código abaixo, podemos observar como criar e exibir variáveis *bool*:

```
logica = False
print(f"A variável contém valor {logica}")
logica = True
print(f"Agora a variável contém valor {logica}")
```

Código-fonte 8 – Criando e exibindo variáveis bool
Fonte: Elaborado pelo autor (2022)

Se o resultado do programa em execução não é muito impressionante, é porque o verdadeiro poder do tipo bool não está em variáveis, mas nas operações lógicas. O quadro abaixo apresenta algumas operações que podemos realizar:

Variações e tipos de dados

Operador	O que retorna?
>	Verdadeiro quando o primeiro valor é maior que o segundo, Falso nos outros casos
<	Verdadeiro quando o primeiro valor é menor que o segundo, Falso nos outros casos
>=	Verdadeiro quando o primeiro valor é maior ou igual ao segundo, Falso nos outros casos
<=	Verdadeiro quando o primeiro valor é menor ou igual ao segundo, Falso nos outros casos
==	Verdadeiro quando o primeiro valor é igual ao segundo, Falso quando são diferentes
!=	Verdadeiro quando o primeiro valor é diferente do segundo, Falso quando são iguais
bool(valor)	Verdadeiro quando o valor é diferente de 0 em variáveis numéricas e diferente de vazio em variáveis de texto, Falso nos outros casos

Quadro 2 – Operadores aritméticos em Python

Fonte: Elaborado pelo autor (2022)

Atualizando o nosso script para testarmos todos esses operadores, temos o seguinte código:

```
logica = False
print(f"A variável contém valor {logica}")
logica = True
print(f"Agora a variável contém valor {logica}")
logica = 1 > 2
print(f"1 é maior que 2? {logica}")
logica = 1 < 2
print(f"1 é menor que 2? {logica}")
logica = 1 >= 2
print(f"1 é maior ou igual que 2? {logica}")
logica = 1 <= 2
print(f"1 é menor ou igual que 2? {logica}")
logica = 1 == 2
print(f"1 é igual a 2? {logica}")
logica = 1 != 2
print(f"1 é diferente de 2? {logica}")
logica = bool(0)
print(f"O valor de 0 convertido em lógico é {logica}")
logica = bool(15)
print(f"O valor de 15 convertido em lógico é {logica}")
```

Código-fonte 9 – Testando todos os operadores bool

Fonte: Elaborado pelo autor (2022)

O tipo `bool` pode não parecer poderoso quando utilizado sozinho, mas será a base do nosso estudo de desvios condicionais mais adiante.

E antes de finalizar este texto, precisamos falar de... texto!

2.4 Eu testo o texto

Desde o início da nossa jornada no mundo do Python estamos utilizando o tipo de dados que armazena texto no Python, o `str`. Este nome refere-se à *string*, que é como chamamos os textos em linguagem de programação.

É isso mesmo: quando você escreveu o seu primeiro `print` (“*Hello World*”), os dados que estavam entre aspas foram considerados do tipo `str` pelo Python, assim como a nossa primeira variável chamada *nome*.

Se o Python é a primeira linguagem de programação que você está aprendendo, talvez não seja possível notar logo de cara, mas para quem já passou por outras linguagens de programação é um verdadeiro júbilo utilizar as `str` em Python.

Assim como os outros tipos de dados, a `str` também tem diversas operações (ou métodos, se formos utilizar o termo mais adequado) disponíveis e a melhor forma de conhecermos e testarmos algumas delas é criando um novo script chamado *tipo_str.py*.

Vamos começar com uma operação simples: escrever um texto que contenha uma quebra de linha e uma tabulação:

```
texto = "Este texto quebra de linha \n aqui. Porém aqui  
\ttemos uma tabulação"  
print(texto)
```

Código-fonte 10 – Criando uma variável de texto com quebra de linha e tabulação
Fonte: Elaborado pelo autor (2022)

Para realizar uma quebra de linha incluímos um `\n` na nossa string, enquanto para incluir uma tabulação utilizamos o `\t`. Estes dois símbolos podem ser utilizados tanto em variáveis quanto em exibições de mensagens, assim como em qualquer lugar onde tenhamos o tipo de dados `str`.

Variações e tipos de dados

Existe uma vasta quantidade de métodos disponíveis nas *str* que pode ser verificada na documentação do Python: <https://docs.python.org/pt-br/3/library/stdtypes.html#textseq>. O código a seguir apresenta apenas algumas delas:

```
#Criação e exibição de uma variável de texto com quebra
de linha e tabulação
texto = "Este texto quebra de linha \n aqui. Porém aqui
\temos uma tabulação"
print(texto)

#O método capitalize() torna o primeira letra do texto
maiúscula
texto = "texto em minusculas AINDA É texto"
print(texto.capitalize())

#O método count() conta quantas vezes um texto aparece
dentro de outro
texto = "texto em minusculas AINDA É texto"
print(texto.count("ex"))

#O método endswith() verifica se um texto termina com
determinada sequencia de caracteres
texto = "texto em minusculas AINDA É texto"
print(texto.endswith("!"))

#O método startswith() verifica se um texto termina com
determinada sequencia de caracteres
texto = "texto em minusculas AINDA É texto"
print(texto.startswith("te"))

#Os métodos upper() e lower() tornam um texto maiúsculo
ou minúsculo, respectivamente
texto = "texto em minusculas AINDA É texto"
print(texto.lower())
print(texto.upper())

#O método replace() substitui uma parte do texto
texto = "texto em minusculas AINDA É texto"
print(texto.replace("texto", "pizza"))

#O operador in verifica se um texto está em outro
texto1 = "texto em minusculas AINDA É texto"
texto2 = "em"
print(texto2 in texto1)
```

Código-fonte 11 – Testando métodos da *str*

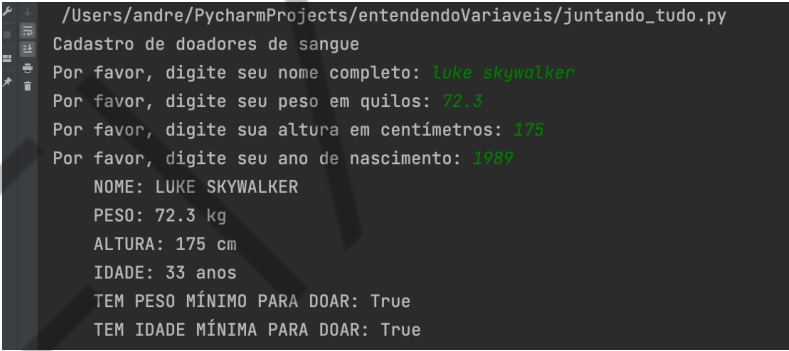
Fonte: Elaborado pelo autor (2022)

Você deve ter reparado que a leitura do código facilita a compreensão de cada método, mas as linhas iniciadas em # não foram exibidas. Isso ocorre porque no Python o # é o símbolo que indica um comentário, ou seja, partes que não devem ser interpretadas como código e que devem ser ignoradas pelo Python. Nós usamos comentários para fazer pequenas anotações nos nossos scripts.

Com comentários ou sem comentários, existe muito a explorar nas variáveis de texto. Por isso vale a pena ler com muita atenção a documentação, fazer diversos testes e imaginar cenários em que você possa utilizar esses recursos.

3 VAMOS JUNTAR TUDO?

Até aqui você acumulou uma bela bagagem no mundo do Python! E este é só o segundo capítulo. Para não perdermos de vista tudo o que estudamos, que tal resolvermos um problema real? Um programa onde o usuário digite o seu nome, seu peso, sua altura e seu ano de nascimento e o programa exiba o seguinte resultado:



```
/Users/andre/PycharmProjects/entendendoVariaveis/juntando_tudo.py
Cadastro de doadores de sangue
Por favor, digite seu nome completo: luke skywalker
Por favor, digite seu peso em quilos: 72.3
Por favor, digite sua altura em centímetros: 175
Por favor, digite seu ano de nascimento: 1989
NOME: LUKE SKYWALKER
PESO: 72.3 kg
ALTURA: 175 cm
IDADE: 33 anos
TEM PESO MÍNIMO PARA DOAR: True
TEM IDADE MÍNIMA PARA DOAR: True
```

Figura 18 – Programa proposto sendo executado
Fonte: Elaborado pelo autor (2022)

Este programa pode ser resolvido de diversas formas. Se você quiser consultar, uma delas está disponível a seguir:

Variações e tipos de dados

```
print("Cadastro de doadores de sangue")
nome = input("Por favor, digite seu nome completo: ")
peso = float(input("Por favor, digite seu peso em quilos: "))
altura = int(input("Por favor, digite sua altura em centímetros: "))
nascimento = int(input("Por favor, digite seu ano de nascimento: "))

idade = 2022 - nascimento
tem_peso_minimo = peso > 50
tem_idade_minima = idade >= 16
texto_saida = f"\tNOME: {nome.upper()}\n\tPESO: {peso} kg\n\tALTURA: {altura} cm\n\tIDADE: {idade} anos\n\tTEM PESO MÍNIMO PARA DOAR: {tem_peso_minimo}\n\tTEM IDADE MÍNIMA PARA DOAR: {tem_idade_minima}\n\t"

print(texto_saida)
```

Código-fonte 12 – Unindo o que estudamos até agora

Fonte: Elaborado pelo autor (2022)

Percebeu o quanto você já aprendeu até aqui? Com os conceitos básicos de Python já conseguimos criar programas que solucionam diversos problemas. Ainda há muitas ferramentas para inserirmos no nosso cinto de utilidades, por isso certifique-se de treinar bastante antes de avançar.

REFERÊNCIAS

LUTZ, M. **Learning Python**. Sebastopol: O'Reilly Media Inc, 2013.

PUGA, S.; RISSETI, G. **Lógica de programação e estrutura de dados com aplicações em Java**. São Paulo: Pearson Prentice Hall, 2009.

EMASP