



---

---

# Oficina Seleção

— Backend com Python —

---

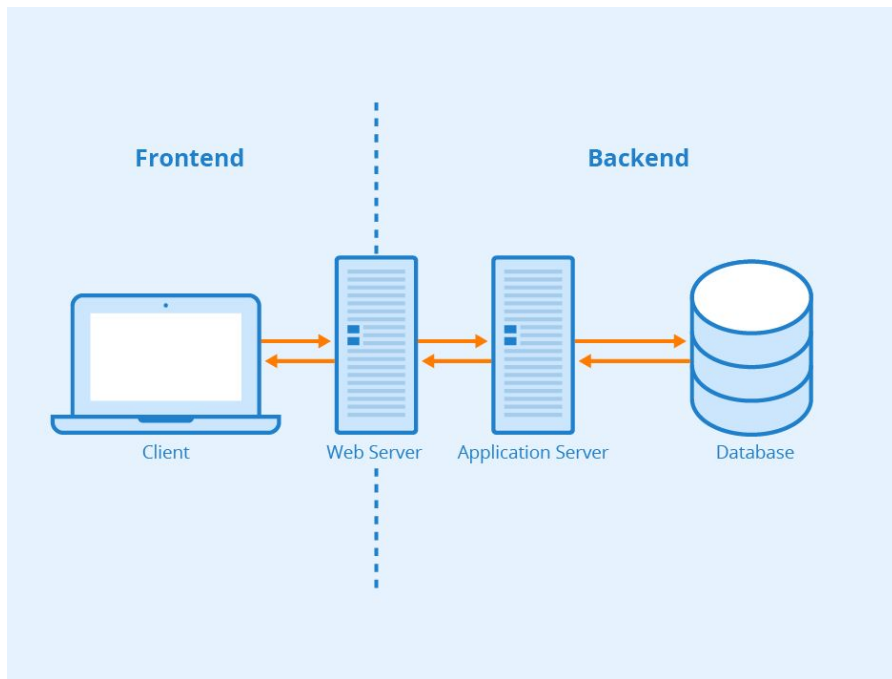
---

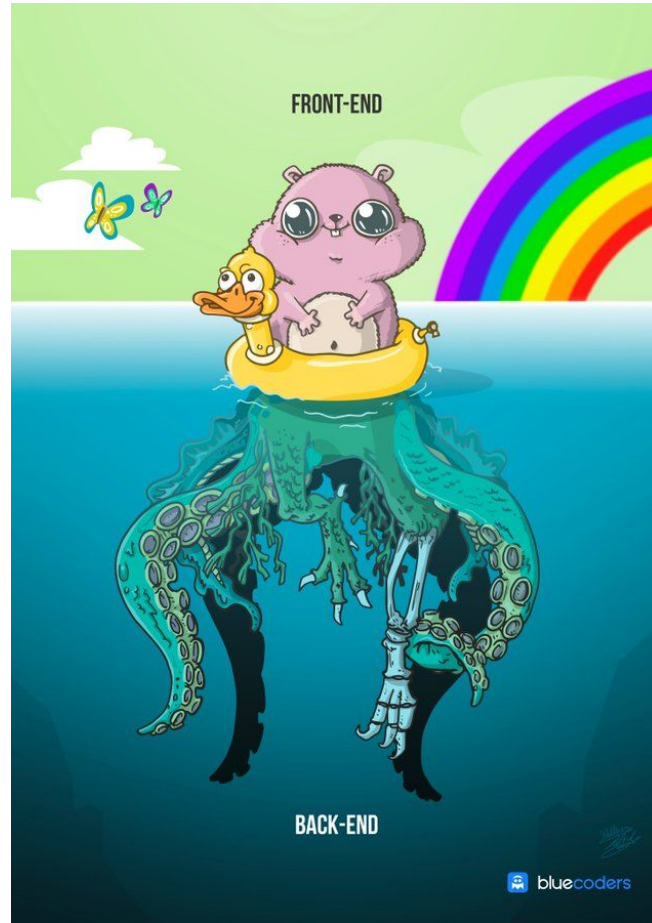


# Pré-requisitos

- Ter o VSCode instalado
- Ter o python instalado
- Vontade de aprender

# Introdução





How the front-end dev sees the project:



How the back-end dev sees the project:



How the project actually is:



What the PM wants:



# Introdução

Porque é importante separar?

Porque aqui focaremos em backend! Sem telas bonitas (por enquanto)

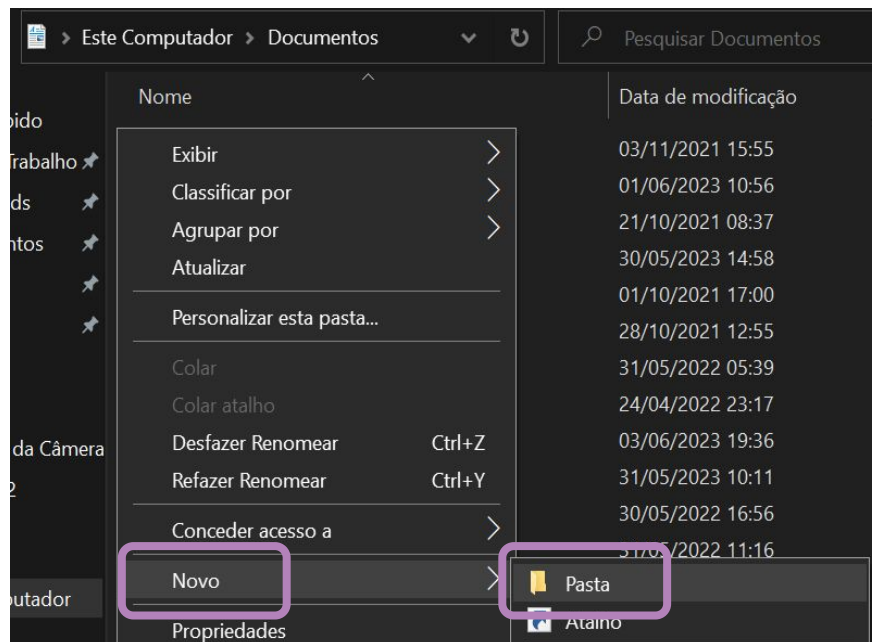


Tá, mas e como eu vou ver o que tô fazendo?



# Tá, mas e como eu vou ver o que tô fazendo?

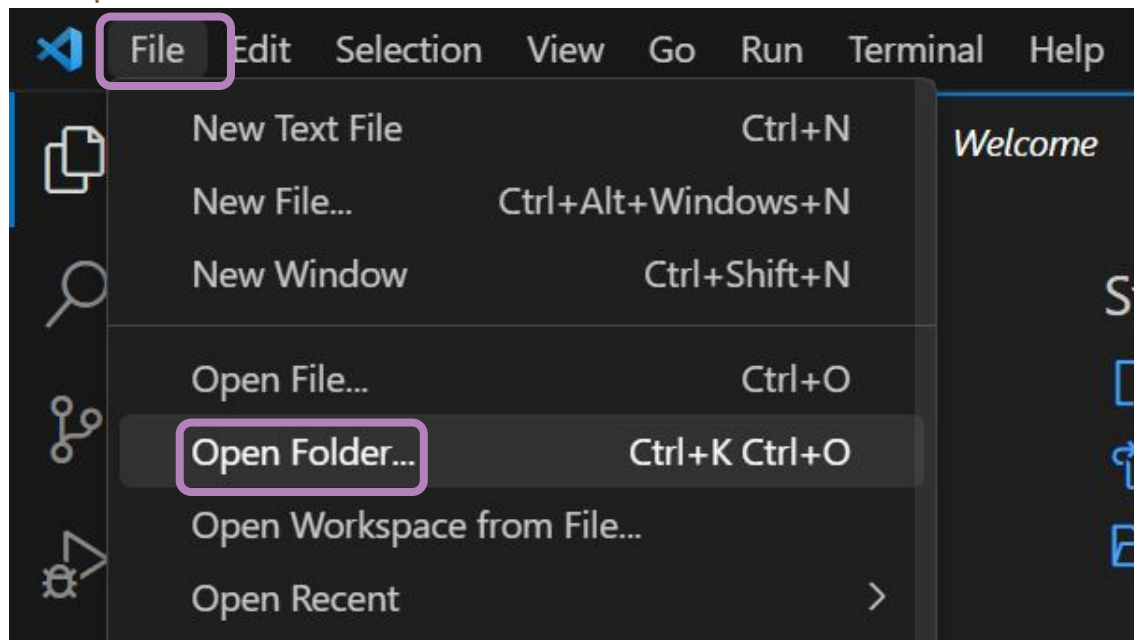
- Abrir a pasta de Documentos do Windows
- Criar uma pasta chamada Reprograma
  - Clicar na pasta com o botão direito
  - Clicar em Novo
  - Clicar em Pasta
  - Digitar Reprograma





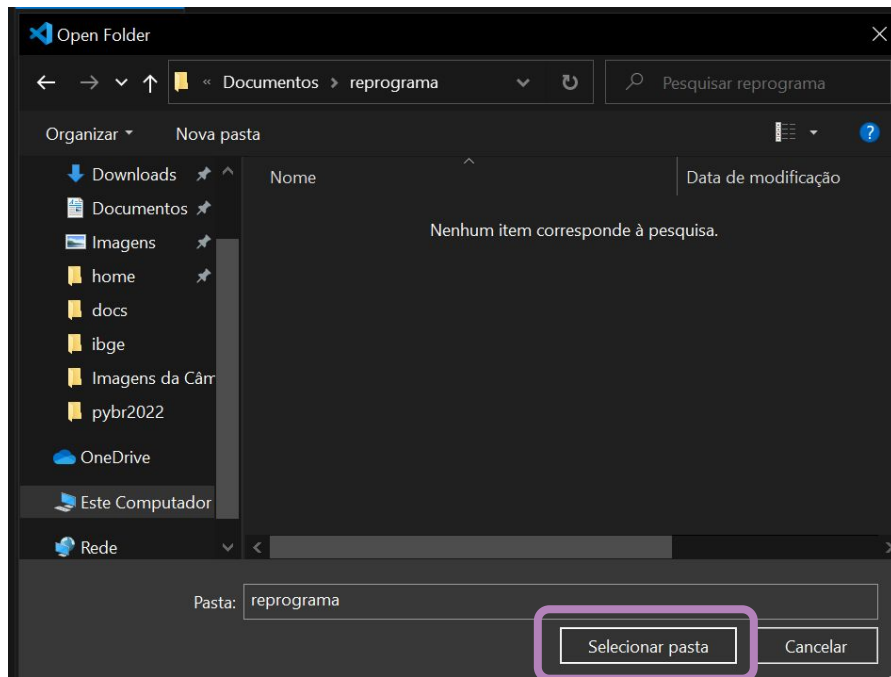
# Abrindo o VSCode

- Abrir o VSCode e abrir a pasta Reprograma
  - Clicar em File ou Arquivo no canto superior esquerdo
  - Clicar em Open Folder ou Abrir Pasta



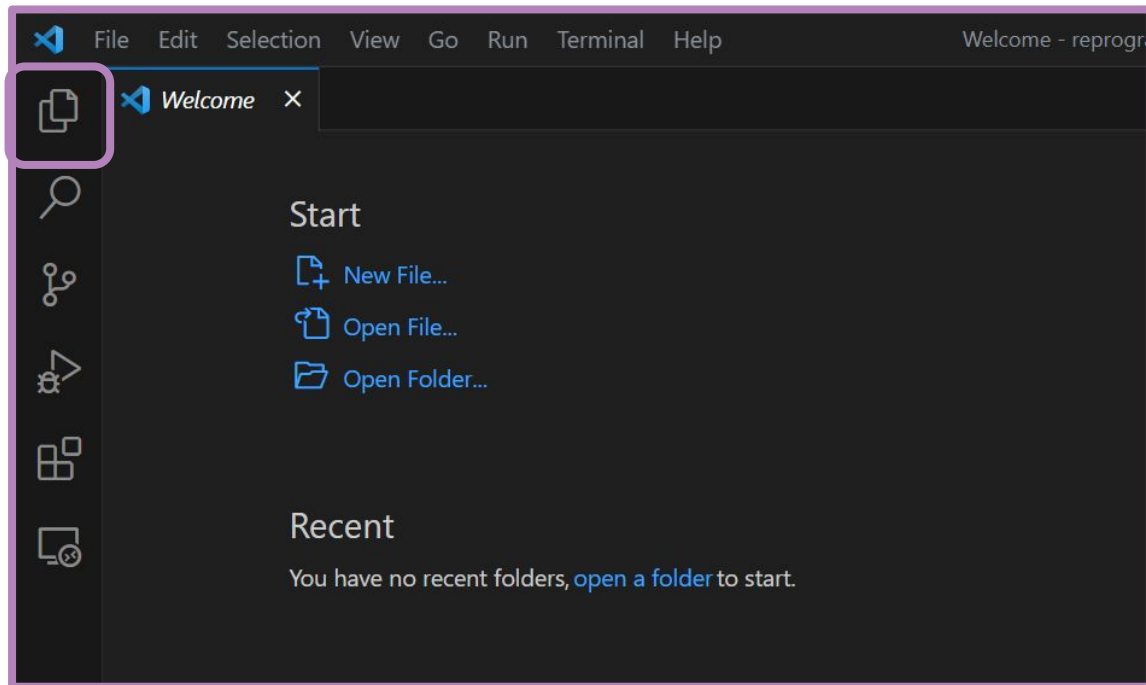
# Abrindo nossa pasta no VSCode

- Clicar a pasta que nós criamos (Reprograma)
- Clicar em Selecionar Pastas



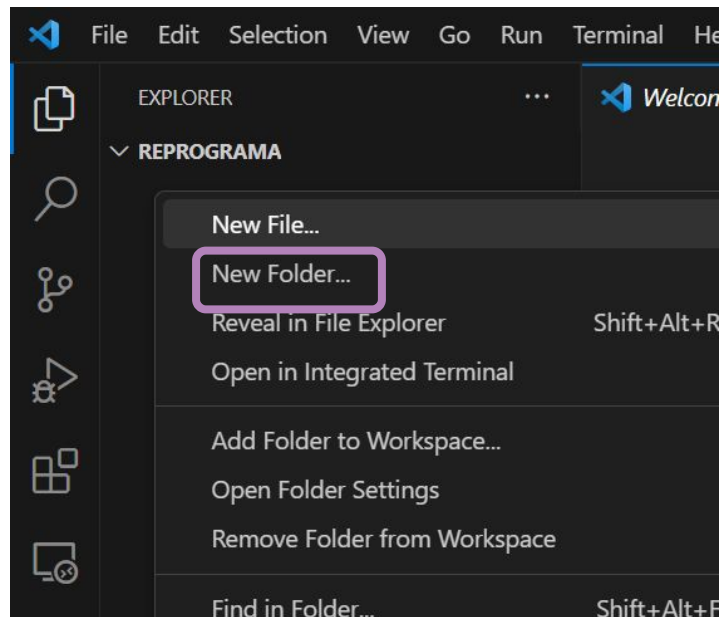
# Criando nosso primeiro arquivo

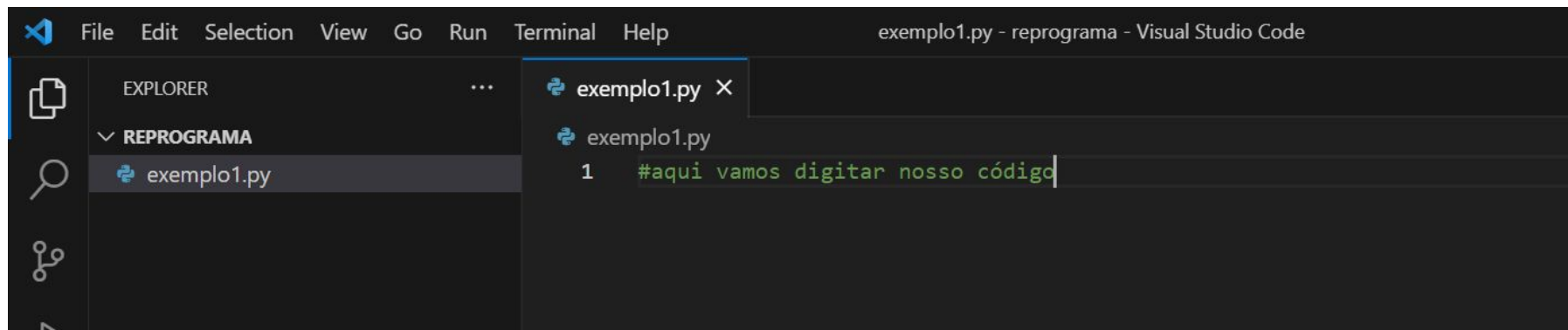
- Abrir o explorador de arquivos



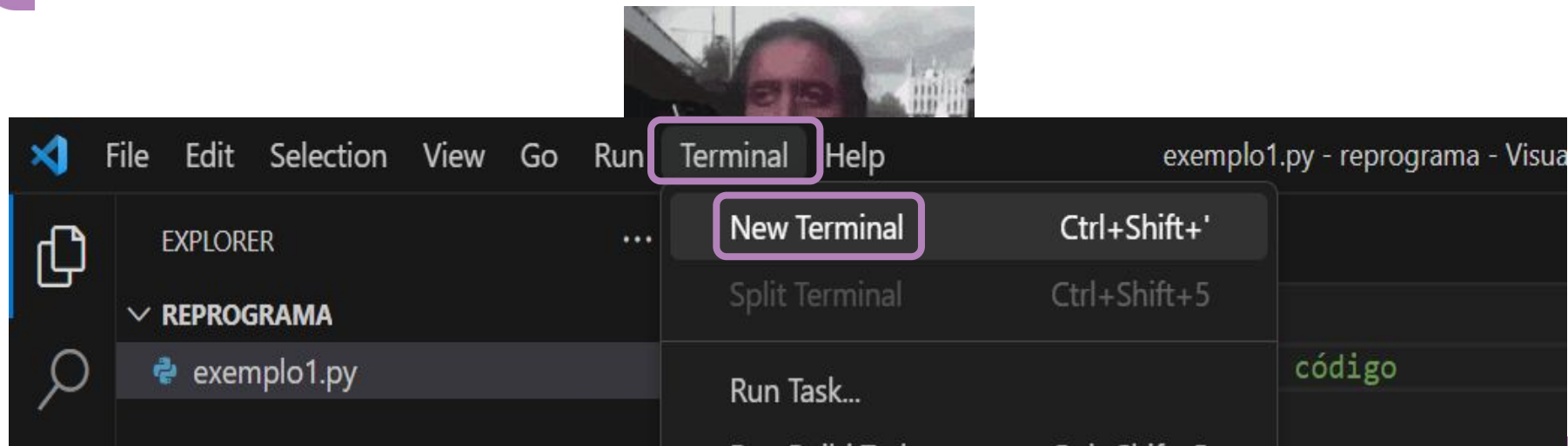
# Criando nosso primeiro arquivo

- Clicar com o botão direito no espaço em branco embaixo do nome da nossa pasta
- Clicar em Novo Arquivo ou New File...
- Digitar o nome do arquivo:
  - exemplo1.py
  - `.py` é a extensão do nosso arquivo
  - Extensão é usada para dizer ao windows o tipo do nosso arquivo





# Agora vamos abrir um terminal



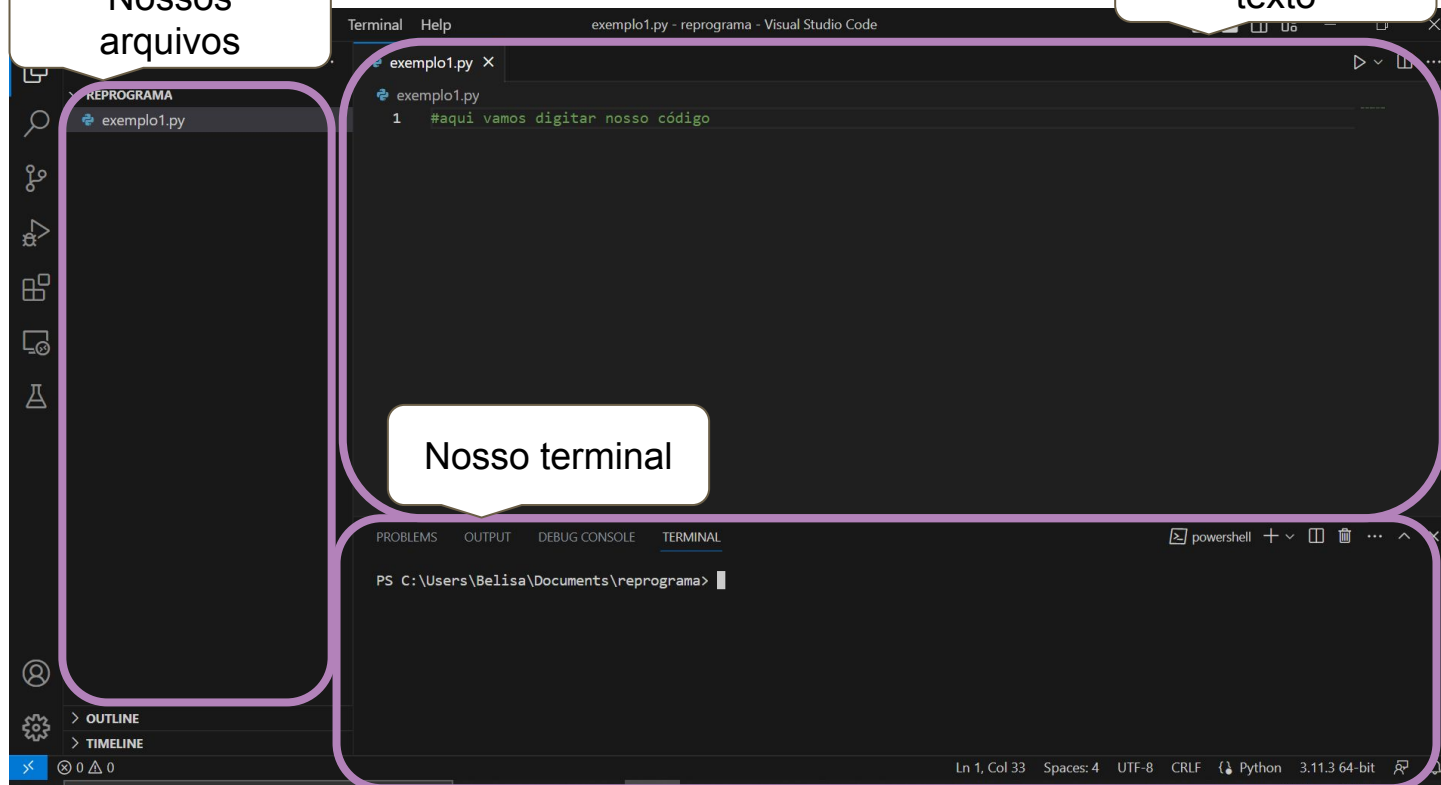
Bem menos assustador do que parece, né?

# Revendo nosso ambiente

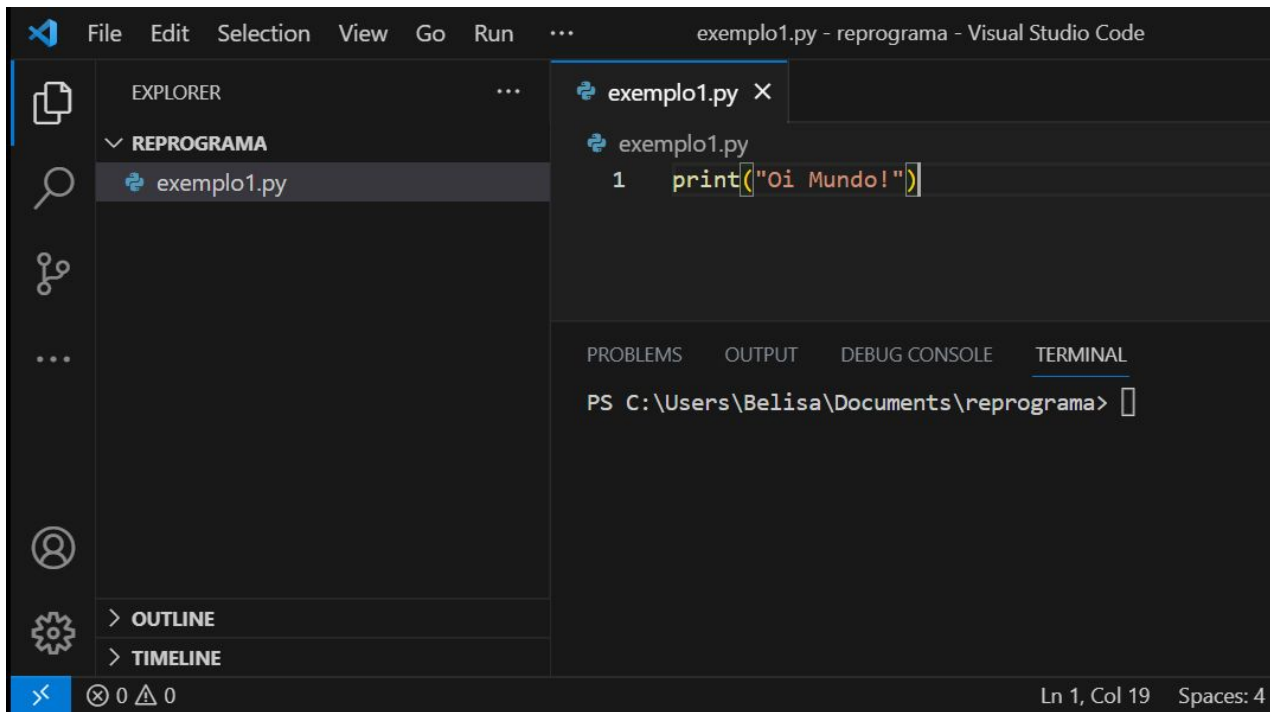
Nossos arquivos

Nosso editor de texto

Nosso terminal

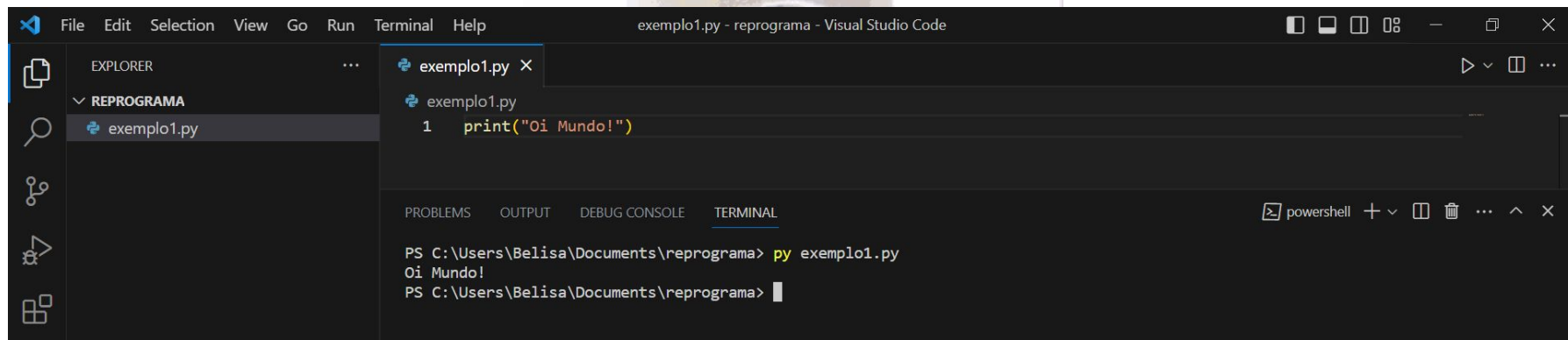


# Vamos escrever nosso primeiro código!





# Bora RODAR rodar nosso primeiro código?



The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left shows a folder named 'REPROGRAMA' containing a file 'exemplo1.py'. The main editor window displays the code for 'exemplo1.py', which consists of a single line: `1 print("Oi Mundo!")`. The bottom panel is divided into 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL' tabs, with 'TERMINAL' selected. The terminal shows the command `PS C:\Users\Belisa\Documents\reprograma> py exemplo1.py` being executed, followed by the output `Oi Mundo!` and a new prompt `PS C:\Users\Belisa\Documents\reprograma>` .

```
File Edit Selection View Go Run Terminal Help
exemplo1.py - reprograma - Visual Studio Code

EXPLORER
  REPROGRAMA
    exemplo1.py

exemplo1.py X
  exemplo1.py
  1 print("Oi Mundo!")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\Belisa\Documents\reprograma> py exemplo1.py
Oi Mundo!
PS C:\Users\Belisa\Documents\reprograma> 
```

**Podemos começar a REprogramar o mundo!**



5 Minutinhos pra comemorar - Tomem água!



# Introdução - Python

“Python é uma linguagem de programação de alto nível, interpretada de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.” - Wikipédia

## Dissecando a definição:

- Linguagem de programação
- Linguagem de programação de alto nível

A língua que a máquina realmente entende é somente 0 e 1. Cada camada de “gíria” que nós colocamos em cima disso, vai tornando a linguagem mais de alto nível. Python é de alto nível, o que quer dizer que ~~ela faz um monte de coisa pra nós~~ nós escrevemos código muito mais próximo da linguagem natural do ser humano, e não dá máquina.

## Dissecando a definição:

- Interpretada de script
- Imperativa, orientada a objetos, funcional

**NA VOLTA  
A GENTE  
COMPRA**

## Dissecando a definição:

- Tipagem dinâmica: quer dizer que uma variável pode trocar seu tipo no meio do rolê
- Tipagem forte: quer dizer que algumas coisas só funcionam para um tipo de variável. Por exemplo, você não pode somar a palavra "a" com o número 1. E você pode somar  $1+1$ , mas não pode somar  $"a" + 1$ .

**PARA! PARA!**  
**PARA! PARA! PARA!**  
**PARA! PARA!**



Variável? Que?!



# Variáveis

- Variável é como uma caixinha em que guardamos nossos dados
- Uma variável tem sempre um nome e seu valor associado
- Nós associamos um valor a um nome através do nosso sinal de atribuição =

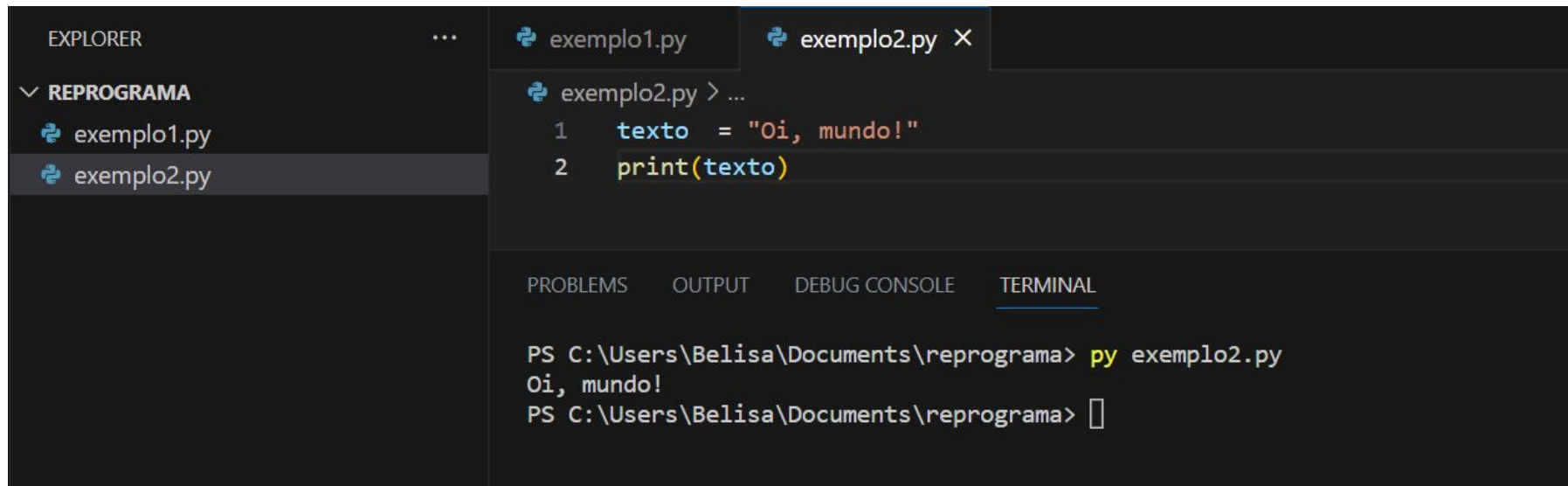
# Criando nossa primeira var



# Criando nossa primeira var - Mão na massa!

- Criar uma variável o nome texto e atribuir a ela o valor “Oi, mundo!”
- Imprimir nossa variável no terminal





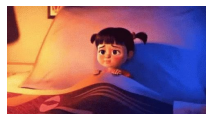
# Entendo variáveis - Mão na massa!

- Criar uma variável o nome numero1 e atribuir a ela um valor numérico
- Imprimir nossa variável no terminal
- Criar uma variável o nome numero2 e atribuir a ela um valor numérico
- Imprimir nossa segunda variável no terminal

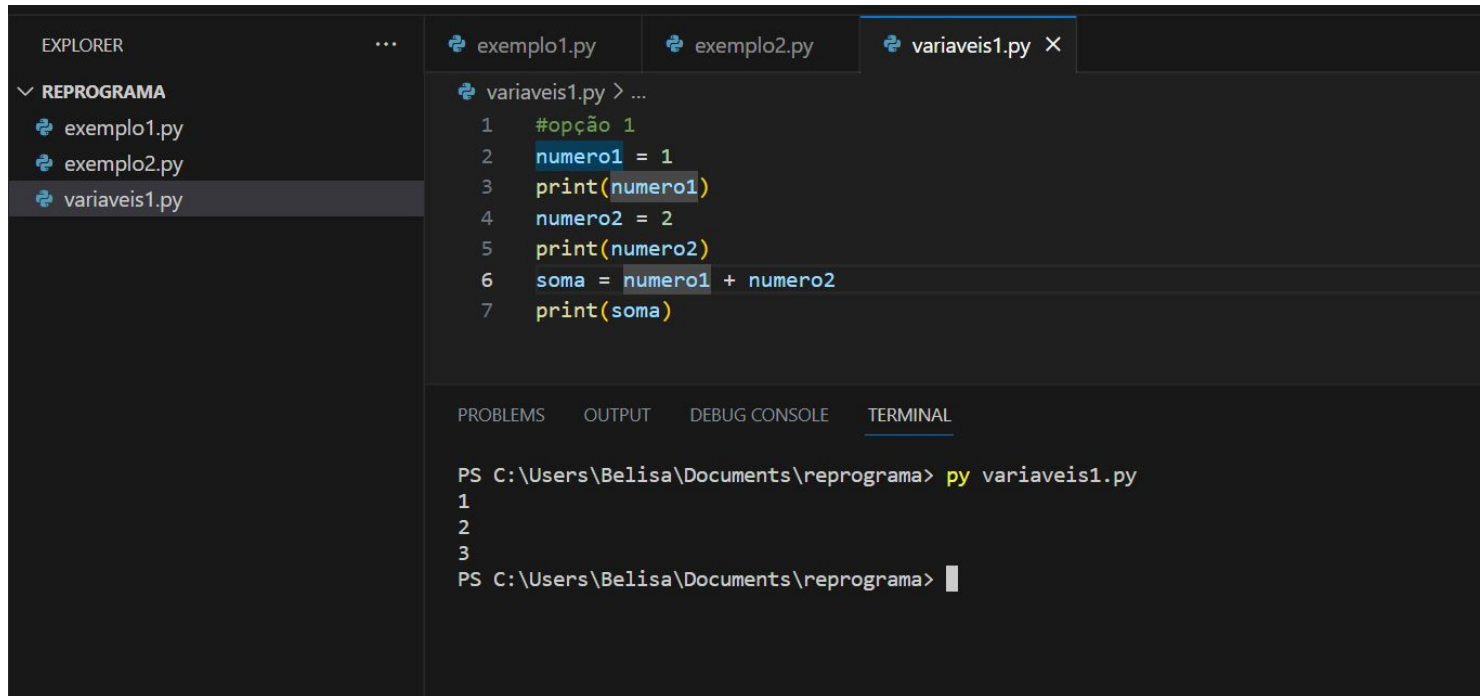


# Entendo variáveis - Mão na massa!

- Criar uma variável o nome numero1 e atribuir a ela um valor numérico
- Imprimir nossa variável no terminal
- Criar uma variável o nome numero2 e atribuir a ela um valor numérico
- Imprimir nossa segunda variável no terminal
- Desafio: somar as duas variáveis e imprimir o resultado
  - Opções:
    - somar as duas variáveis em uma variável chamada soma
    - somar as duas variáveis sobrescrevendo o valor de uma delas
    - somar diretamente na impressão

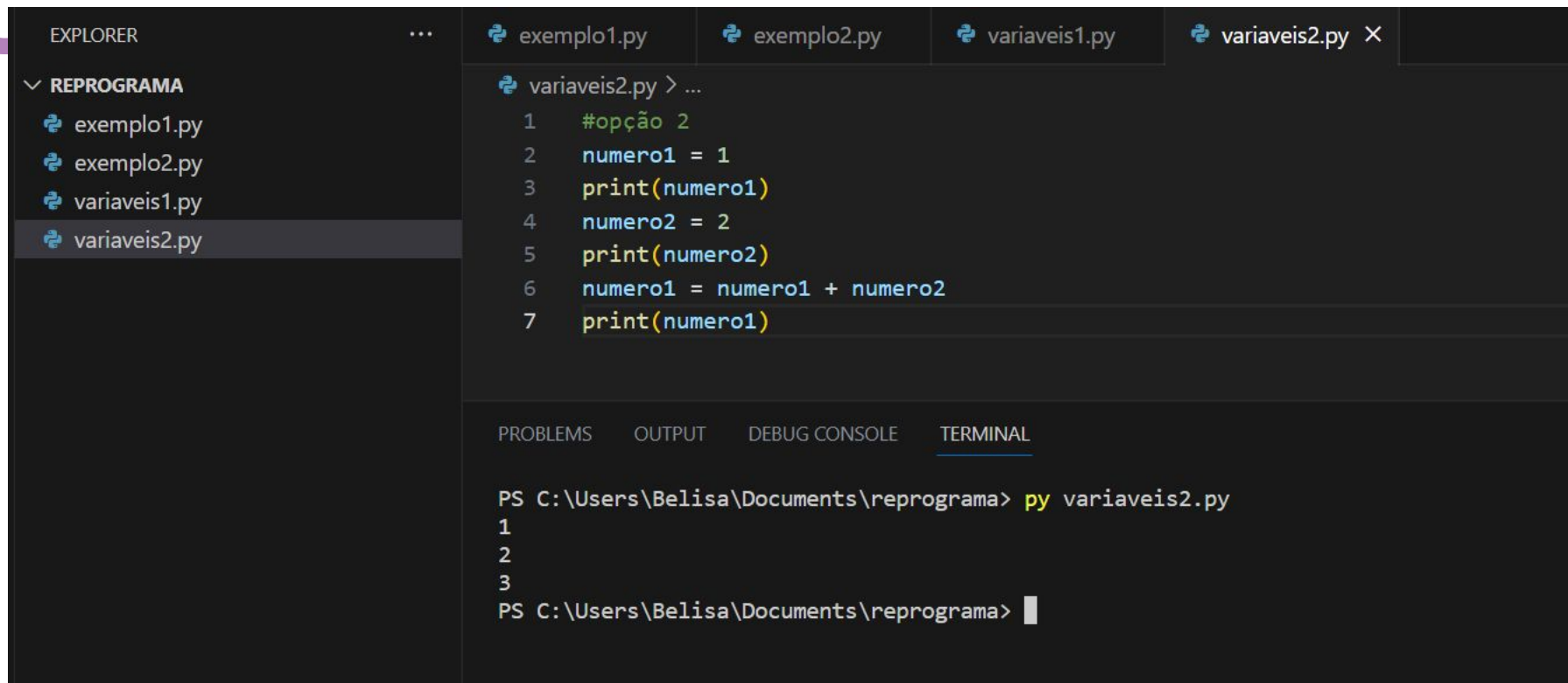


# Opção 1



The image shows a Visual Studio Code interface with a dark theme. On the left, the 'EXPLORER' sidebar shows a folder named 'REPROGRAMA' containing three Python files: 'exemplo1.py', 'exemplo2.py', and 'variaveis1.py'. The 'variaveis1.py' file is selected. The main editor area displays the code for 'variaveis1.py', which consists of seven lines of Python code. The code defines two variables, 'numero1' and 'numero2', prints their values, calculates their sum, and prints the result. The bottom panel shows the 'TERMINAL' tab, which contains the command to run the script and its output.

```
EXPLORER  
  ...  
  REPROGRAMA  
    exemplo1.py  
    exemplo2.py  
    variaveis1.py  
  
  variaveis1.py > ...  
  1  #opção 1  
  2  numero1 = 1  
  3  print(numero1)  
  4  numero2 = 2  
  5  print(numero2)  
  6  soma = numero1 + numero2  
  7  print(soma)  
  
  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  
  
  PS C:\Users\Belisa\Documents\reprograma> py variaveis1.py  
  1  
  2  
  3  
  PS C:\Users\Belisa\Documents\reprograma> 
```



The image shows a Visual Studio Code interface with a dark theme. On the left, the 'EXPLORER' sidebar shows a folder named 'REPROGRAMA' containing four Python files: 'exemplo1.py', 'exemplo2.py', 'variaveis1.py', and 'variaveis2.py'. The file 'variaveis2.py' is selected and highlighted. The main editor area displays the code for 'variaveis2.py' with line numbers 1 through 7. The code defines two variables, 'numero1' and 'numero2', prints their values, and then updates 'numero1' to be the sum of both. Below the code editor, the 'TERMINAL' tab is active, showing the command 'py variaveis2.py' being executed in a PowerShell prompt. The output of the script is displayed as three lines: '1', '2', and '3'.

```
EXPLORER
```

... exemplo1.py exemplo2.py variaveis1.py variaveis2.py X

✓ REPROGRAMA

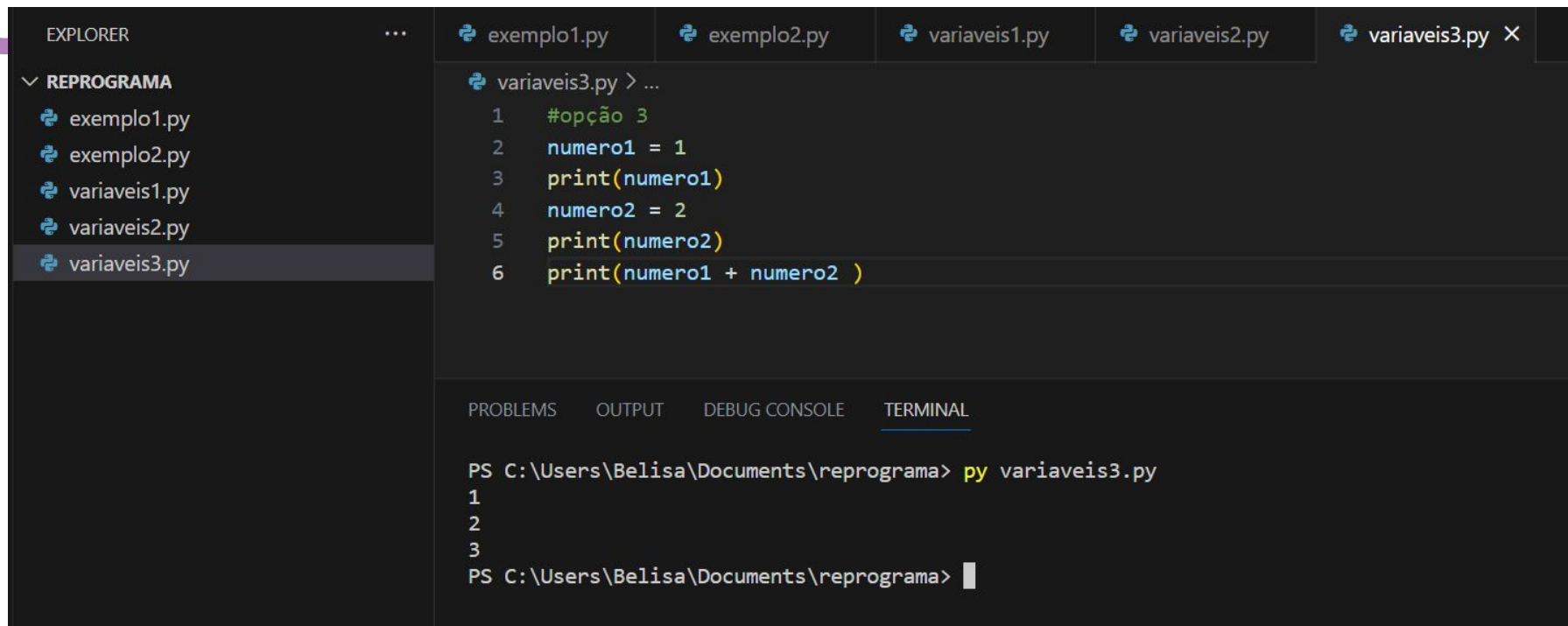
- exemplo1.py
- exemplo2.py
- variaveis1.py
- variaveis2.py

```
variaveis2.py > ...
1  #opção 2
2  numero1 = 1
3  print(numero1)
4  numero2 = 2
5  print(numero2)
6  numero1 = numero1 + numero2
7  print(numero1)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
PS C:\Users\Belisa\Documents\reprograma> py variaveis2.py
1
2
3
PS C:\Users\Belisa\Documents\reprograma> 
```





# 5 Minutinhos de alegria - Tomem água!



# Tipos

[docs.python.org/pt-br/3/library/index.html](https://docs.python.org/pt-br/3/library/index.html)

- selecionad...

or

o Completa

O

- Tipos numéricos — `int`, `float`, `complex`
- ~~Tipos iteradores~~
- ~~Tipos sequências~~ — `list`, `tuple`, `range`
- Tipo sequência de texto — `str`
- ~~Tipos de Sequência Binária~~ — `bytes`, `bytearray`, `memoryview`
- ~~Tipo conjuntos~~ — `set`, `frozenset`
- ~~Tipo mapeamento~~ — `dict`

# Tipos - Numéricos

- Tipos numéricos — `int`, `float`, `complex`

Tipos numéricos representam, bem, números



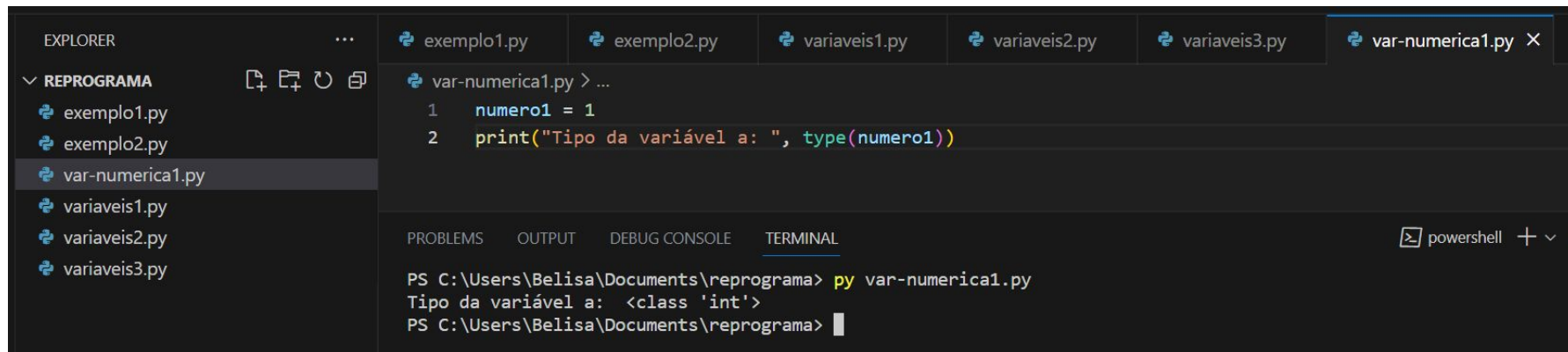
Mas porque 3 tipos?

## Tipos - Numéricos

- Quanto mais “simples” o número, menos memória ele ocupa.
- Mas muitas vezes nós precisamos de números com mais informações.
- O número é inteiro e positivo?
- O número é fracionário?
- O número ~~tá precisando uma terapia~~ é complexo?
  - Neste momento, não se preocupem com complexos. Vamos focar em float e int.

# Tipos - Numéricos - Inteiros

- No python, nós não dizemos explicitamente o tipo da variável que estamos criando. Então vamos usar a função `type` para saber o tipo da nossa variável:



The screenshot shows a Python IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'REPROGRAMA' with several Python files. The code editor shows a file named 'var-numerica1.py' with the following code:

```
1 numero1 = 1
2 print("Tipo da variável a: ", type(numero1))
```

Below the code editor, there is a terminal window showing the output of the script:

```
PS C:\Users\Belisa\Documents\reprograma> py var-numerica1.py
Tipo da variável a: <class 'int'>
PS C:\Users\Belisa\Documents\reprograma>
```

# Tipos - Numéricos - Inteiros

- O que acontece se informamos um número MUITO grande?

The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar lists files under the 'REPROGRAMA' folder: exemplo1.py, exemplo2.py, var-numerica1.py (selected), variaveis1.py, variaveis2.py, and variaveis3.py. The main editor area displays the contents of 'var-numerica1.py', which contains two lines of Python code:

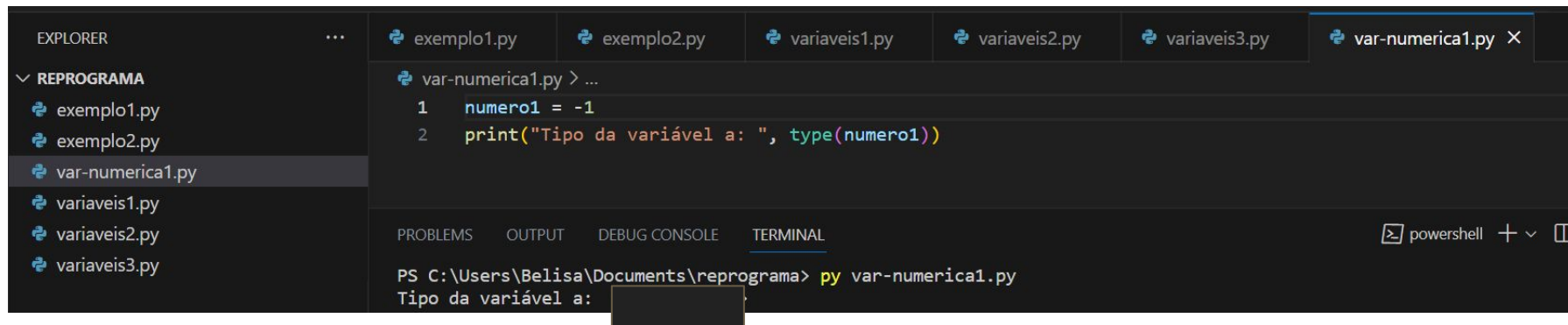
```
1 numero1 = 10000000000000000000  
2 print("Tipo da variável a: ", type(numero1))
```

Below the editor, the TERMINAL panel is active, showing the command prompt output:

```
PS C:\Users\Belisa\Documents\reprograma> py var-numerica1.py  
Tipo da variável a:
```

# Tipos - Numéricos - Inteiros

- O que acontece se informamos um número negativo?



The screenshot shows the Visual Studio Code interface. On the left, the EXPLORER pane displays a folder named 'REPROGRAMA' containing several Python files: exemplo1.py, exemplo2.py, var-numerica1.py (selected), variaveis1.py, variaveis2.py, and variaveis3.py. The main editor area shows the content of 'var-numerica1.py', which contains two lines of code: `1 numero1 = -1` and `2 print("Tipo da variável a: ", type(numero1))`. Below the editor, the TERMINAL pane is active, showing the command prompt 'PS C:\Users\Belisa\Documents\reprograma> py var-numerica1.py' and the output 'Tipo da variável a: ' followed by a small black rectangular box, indicating the output of the `type()` function for a negative integer.

```
EXPLORER
```

... exemplo1.py exemplo2.py variaveis1.py variaveis2.py variaveis3.py var-numerica1.py X

REPROGRAMA

- exemplo1.py
- exemplo2.py
- var-numerica1.py
- variaveis1.py
- variaveis2.py
- variaveis3.py

```
var-numerica1.py > ...  
1 numero1 = -1  
2 print("Tipo da variável a: ", type(numero1))
```

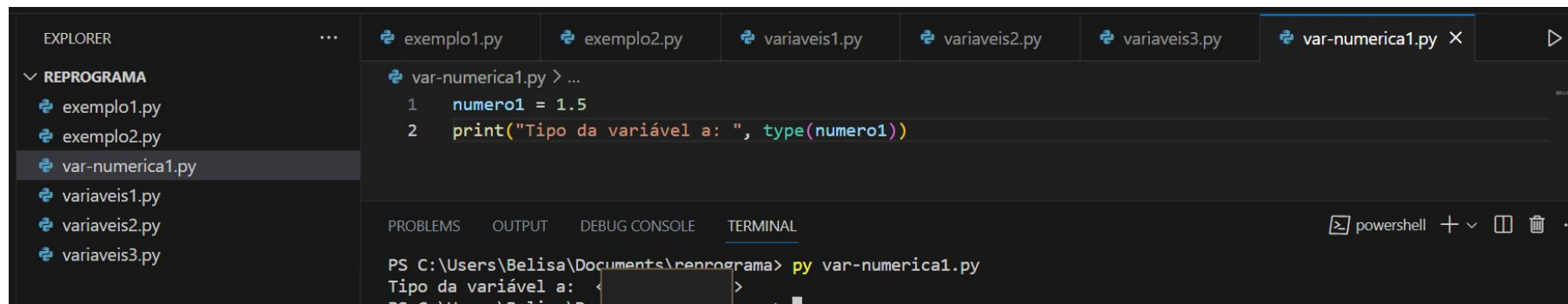
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\Belisa\Documents\reprograma> py var-numerica1.py  
Tipo da variável a:



# Tipos - Numéricos - Ponto Flutuante (Float)

- O que acontece se informamos um número com parte fracionária? (com vírgula)



The screenshot shows a Python IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'REPROGRAMA' with several Python files. The code editor shows a file named 'var-numerica1.py' with the following code:

```
var-numerica1.py > ...  
1 numero1 = 1.5  
2 print("Tipo da variável a: ", type(numero1))
```

The terminal at the bottom shows the command to run the script and the output:

```
PS C:\Users\Belisa\Documents\reprograma> py var-numerica1.py  
Tipo da variável a: <float>
```

## Tipos - Numéricos - Ponto Flutuante (Float)

- O nome ponto flutuante vem da ideia de que um número por ter uma ou mais casas decimais e uma ou mais casas fracionárias, então nós precisamos que o ponto “flutue” para ficar no local correto.

```
b = 0.500009
```

```
b = 100000.500009
```

```
b = 100000.5
```

# Tipos - Numéricos - Tipagem dinâmica

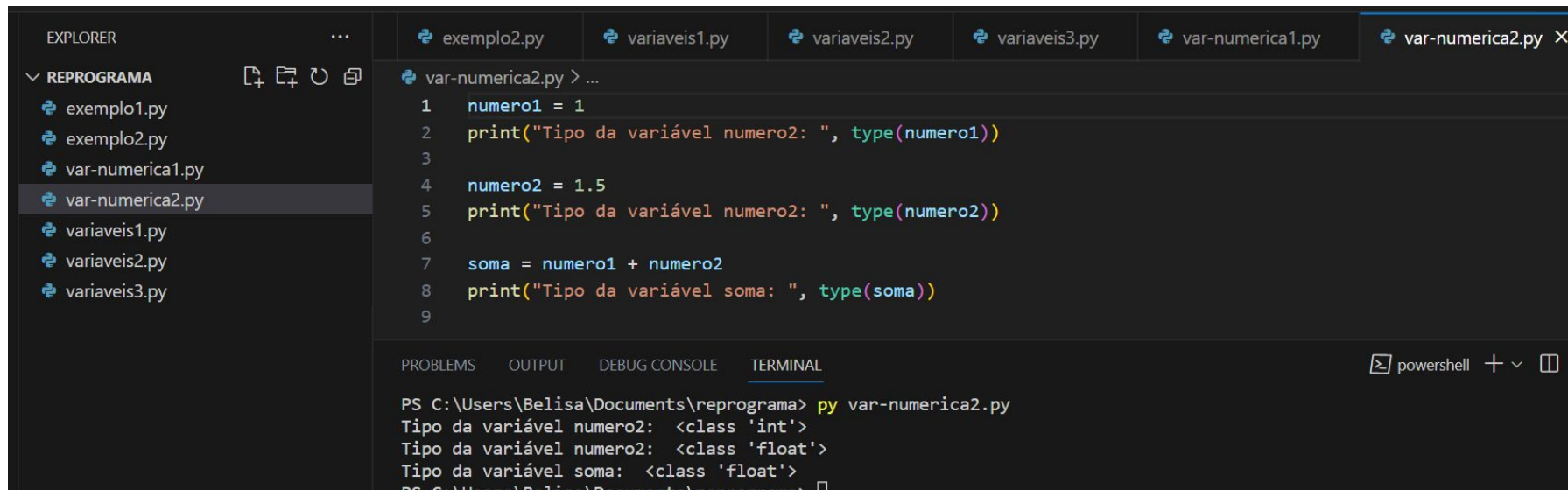
- O que acontece quando somamos números de tipos diferentes?
- O programa abaixo executa sem erros?
- Se sim, qual o tipo da variável soma?

```
var-numerica2.py > ...
1  numero1 = 1
2  print("Tipo da variável numero2: ", type(numero1))
3
4  numero2 = 1.5
5  print("Tipo da variável numero2: ", type(numero2))
6
7  soma = numero1 + numero2
8  print("Tipo da variável soma: ", type(soma))
9
```

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

# Tipos - Numéricos - Tipagem dinâmica

- Somente é possível ter o valor completo se o tipo permitir um valores fracionários



The screenshot shows a Python IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'REPROGRAMA' with several Python files. The file 'var-numerica2.py' is selected. The code editor shows the following Python code:

```
1 numero1 = 1
2 print("Tipo da variável numero2: ", type(numero1))
3
4 numero2 = 1.5
5 print("Tipo da variável numero2: ", type(numero2))
6
7 soma = numero1 + numero2
8 print("Tipo da variável soma: ", type(soma))
9
```

Below the code editor is a terminal window showing the output of the script:

```
PS C:\Users\Belisa\Documents\reprograma> py var-numerica2.py
Tipo da variável numero2: <class 'int'>
Tipo da variável numero2: <class 'float'>
Tipo da variável soma: <class 'float'>
```

# Tipos - Sequência de texto

- Tipo sequência de texto — `str`

- Variáveis do tipo `str` (ou `string`) são variáveis de texto
- São chamadas de sequência de texto porque o python entende texto caracter (cada letra, símbolo, espaço...) por caracter.
- Variáveis “string” devem ser informadas entre aspas.

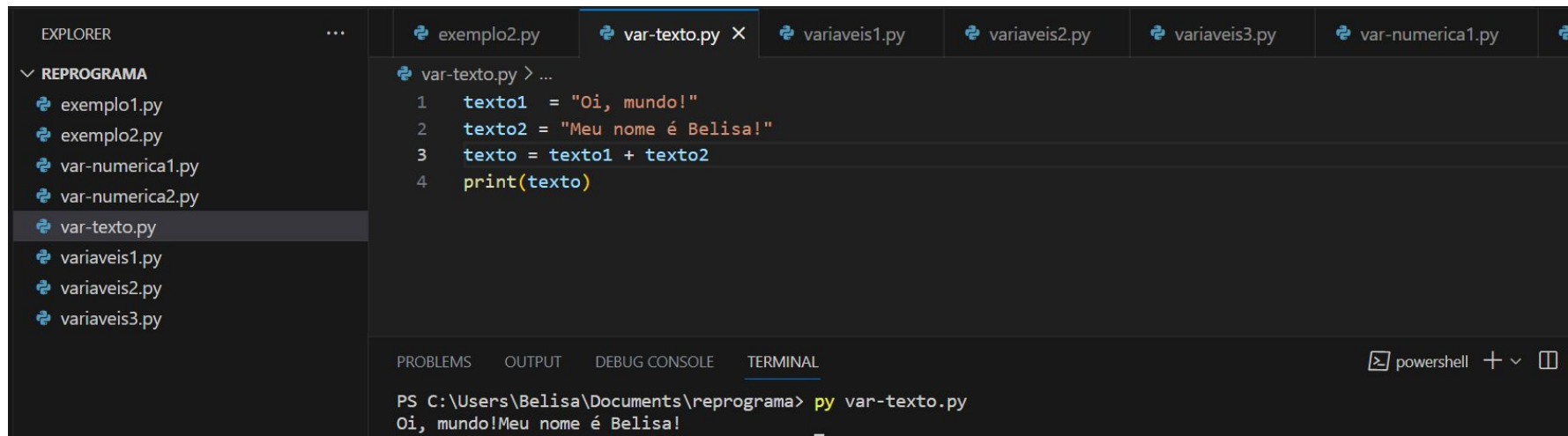
## Tipos - Sequência de texto - Mão na massa

- Cria uma variável chamada texto com o valor "Oi, mundo!" e imprimir essa variável



# Tipos - Sequência de texto - Mão na massa

- Criar duas variáveis de texto e imprimir elas juntas
  - Como o python trata texto como uma sequência, nós podemos SOMAR duas sequências de texto



The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a folder named 'REPROGRAMA' containing several Python files. The file 'var-texto.py' is selected and its code is displayed in the editor. The code defines two string variables, 'texto1' and 'texto2', concatenates them into 'texto', and prints the result. The terminal shows the command to run the script and the output, which is the concatenated string.

```
EXPLORER
```

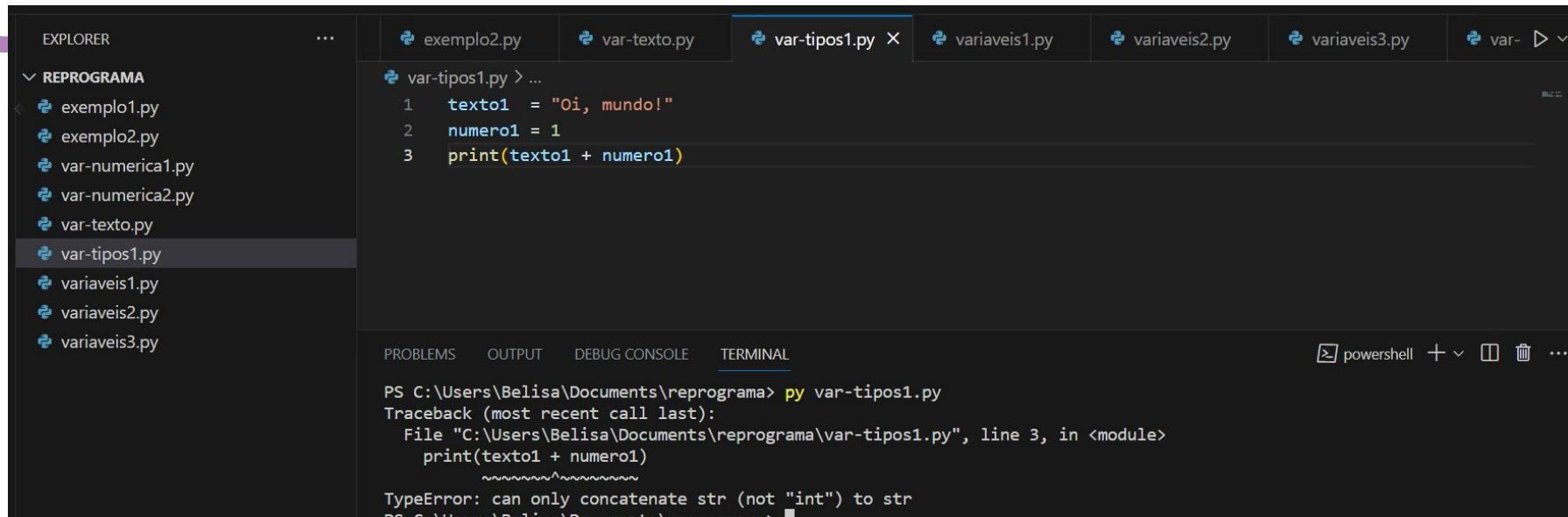
- REPROGRAMA
  - exemplo1.py
  - exemplo2.py
  - var-numerica1.py
  - var-numerica2.py
  - var-texto.py
  - variaveis1.py
  - variaveis2.py
  - variaveis3.py

```
var-texto.py > ...
1  texto1 = "Oi, mundo!"
2  texto2 = "Meu nome é Belisa!"
3  texto = texto1 + texto2
4  print(texto)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
```

```
PS C:\Users\Belisa\Documents\reprograma> py var-texto.py
Oi, mundo!Meu nome é Belisa!
```

# Porque entender tipos?



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left lists files under the 'REPROGRAMA' folder, with 'var-tipos1.py' selected. The main editor displays the contents of 'var-tipos1.py':

```
var-tipos1.py > ...
1 texto1 = "Oi, mundo!"
2 numero1 = 1
3 print(texto1 + numero1)
```

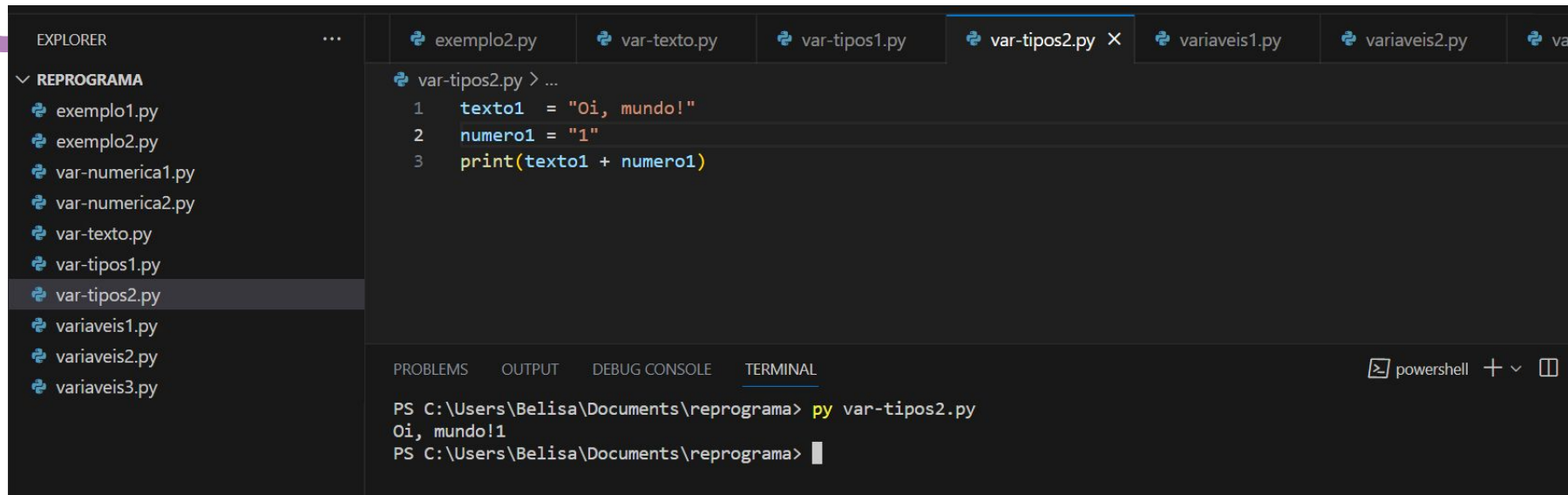
The TERMINAL panel at the bottom shows the command 'py var-tipos1.py' and the resulting error message:

```
PS C:\Users\Belisa\Documents\reprograma> py var-tipos1.py
Traceback (most recent call last):
  File "C:\Users\Belisa\Documents\reprograma\var-tipos1.py", line 3, in <module>
    print(texto1 + numero1)
          ~~~~~^~~~~~
TypeError: can only concatenate str (not "int") to str
PS C:\Users\Belisa\Documents\reprograma>
```

Erro de tipo: só é possível concatenar str (não "int") com str :(



# Agora sim! Mas o que mudou?



The image shows a Visual Studio Code interface with a dark theme. On the left, the 'EXPLORER' sidebar shows a project named 'REPROGRAMA' with several Python files. The file 'var-tipos2.py' is selected and highlighted. The main editor area displays the contents of 'var-tipos2.py', which contains three lines of Python code. Below the editor, the 'TERMINAL' panel is active, showing the command to run the script and its output.

```
EXPLORER
```

... exemplo2.py var-texto.py var-tipos1.py var-tipos2.py variaveis1.py variaveis2.py va

✓ REPROGRAMA

- exemplo1.py
- exemplo2.py
- var-numerica1.py
- var-numerica2.py
- var-texto.py
- var-tipos1.py
- var-tipos2.py
- variaveis1.py
- variaveis2.py
- variaveis3.py

```
var-tipos2.py > ...
1 texto1 = "Oi, mundo!"
2 numero1 = "1"
3 print(texto1 + numero1)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell + v □

```
PS C:\Users\Belisa\Documents\reprograma> py var-tipos2.py
Oi, mundo!1
PS C:\Users\Belisa\Documents\reprograma> 
```

# Variáveis e tipagem - O que aconteceu aqui?



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane lists files in a folder named 'REPROGRAMA'. The file 'var-tipos3.py' is selected. The main editor area shows the code for 'var-tipos3.py':

```
1 texto1 = "Oi, mundo!"  
2 numero1 = 1  
3 numero1 = str(numero1)  
4 print(texto1 + numero1)
```

Below the editor, the TERMINAL pane is active, showing the command prompt output:

```
PS C:\Users\Belisa\Documents\reprograma> py var-tipos3.py  
Oi, mundo!1  
PS C:\Users\Belisa\Documents\reprograma>
```

Porque agora funcionou?

5 Minutinhos pra refletir - Tomem água!



# Funções em python!

Ok, não podemos somar "a"+1 mas podemos juntar a palavra "a" com a **palavra** "1": "a1". Então para juntar "a" + 1, nós precisamos primeiro dizer para o python que 1, na verdade, é uma palavra: "1".

```
b = str(b)
```

A *função* str() transforma uma variável de um *tipo* para outro tipo, no caso, string.

# Funções em python - Definição

Uma série de instruções que retorna algum valor para um chamador.  
Também pode ser passado zero ou mais *argumentos* que podem ser usados na execução do corpo. - Documentação Python

# Funções em python - Exemplos - Mão na massa

- `print(var)`
- `str(var)`
- `int(var)`
- `type(var)`
- `print(var1, var2)`
- `max(var1, var2)`
- `min(var1, var2)`



# Ferramentas de controle de fluxo - Condicionais

- Agora nós já sabemos usar variáveis e manipular elas com funções.
- Vamos dar o próximo passo: manipular variáveis só em alguns casos.
- Para isso, usaremos a instrução **if**, que em português significa “se”
- Essa instrução recebe um valor que deve ser avaliado.
- Se o valor for verdadeiro, então ela executa o que estiver no bloco interno (calma, o bloco interno já já vai fazer sentido!)
- Depois de executar (ou não!) o bloco interno, segue executando os demais comandos normalmente

# Ferramentas de controle de fluxo - Condicionais

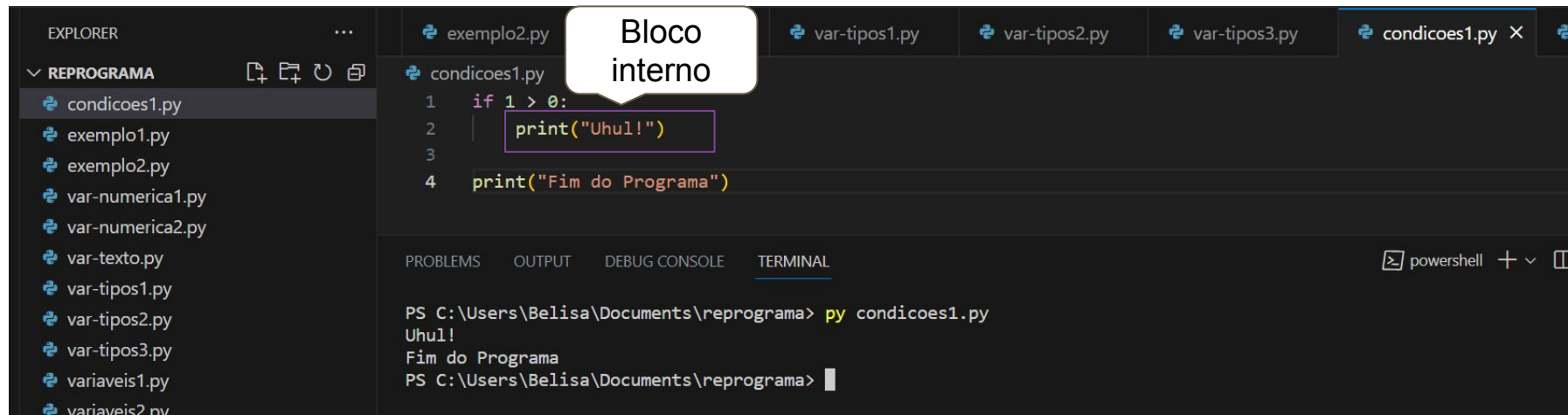
- Para o primeiro exemplo, vamos usar uma expressão que sempre vai ser verdadeira.
- Vamos testar se 1 é maior do que 0 através do sinal de comparação da matemática: > (maior que)

Se a **condição** é **verdadeira**, imprime "Uhul!"

Se 1 é **maior** que 0, imprime "Uhul!"



# Ferramentas de controle de fluxo - Condicionais



The screenshot shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a folder named 'REPROGRAMA' containing several Python files. The file 'condicoes1.py' is selected. The code editor shows the following Python code:

```
1 if 1 > 0:
2     print("Uhu!")
3
4 print("Fim do Programa")
```

A callout box labeled 'Bloco interno' points to the indented code block (lines 2-3). The terminal at the bottom shows the command 'py condicoes1.py' being executed, resulting in the output 'Uhu!' and 'Fim do Programa'.

# Ferramentas de controle de fluxo - Condicionais

## Operadores Relacionais

Porque não  
usar somente  
um "="?

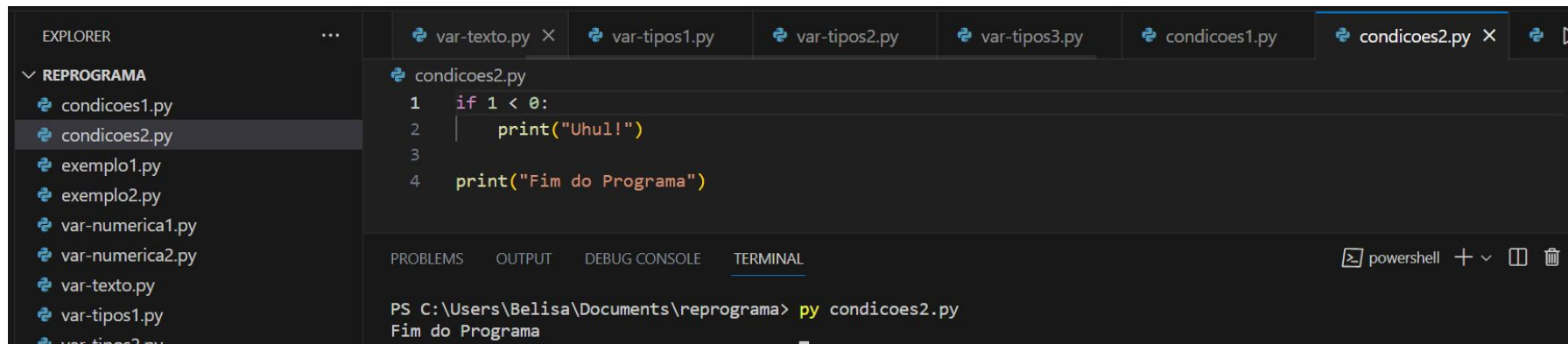
> Maior	== Igualdade
< Menor	<> Diferente
<= Menor ou igual	!= Diferente
>= Maior ou igual	

# Ferramentas de controle de fluxo - Condicionais

- Agora vamos fazer o oposto, vamos usar uma expressão que sempre vai ser falsa.
- Vamos testar se 1 é menor do que 0 através do sinal de comparação da matemática:  $<$  (menor que)

Se 1 é **menor** que 0, imprime "Uhul!"

# Ferramentas de controle de fluxo - Condicionais



The screenshot displays the Visual Studio Code interface. On the left, the Explorer sidebar shows a folder named 'REPROGRAMA' containing several Python files. The file 'condicoes2.py' is selected and open in the main editor. The code in the editor is a simple conditional statement in Python. Below the editor, the TERMINAL panel is active, showing the command to run the script and its output.

```
EXPLORER
```

- REPROGRAMA
  - condicoes1.py
  - condicoes2.py
  - exemplo1.py
  - exemplo2.py
  - var-numerica1.py
  - var-numerica2.py
  - var-texto.py
  - var-tipos1.py
  - var-tipos2.py

```
condicoes2.py
1  if 1 < 0:
2      print("Uhu!")
3
4  print("Fim do Programa")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

powershell + - [ ] [X]

```
PS C:\Users\Belisa\Documents\reprograma> py condicoes2.py
Fim do Programa
```

# Ferramentas de controle de fluxo - Condicionais

- Porque nosso “Uhul!” não foi impresso no terminal?
- Porque 1 é maior que 0, então todo o bloco interno é ignorado.

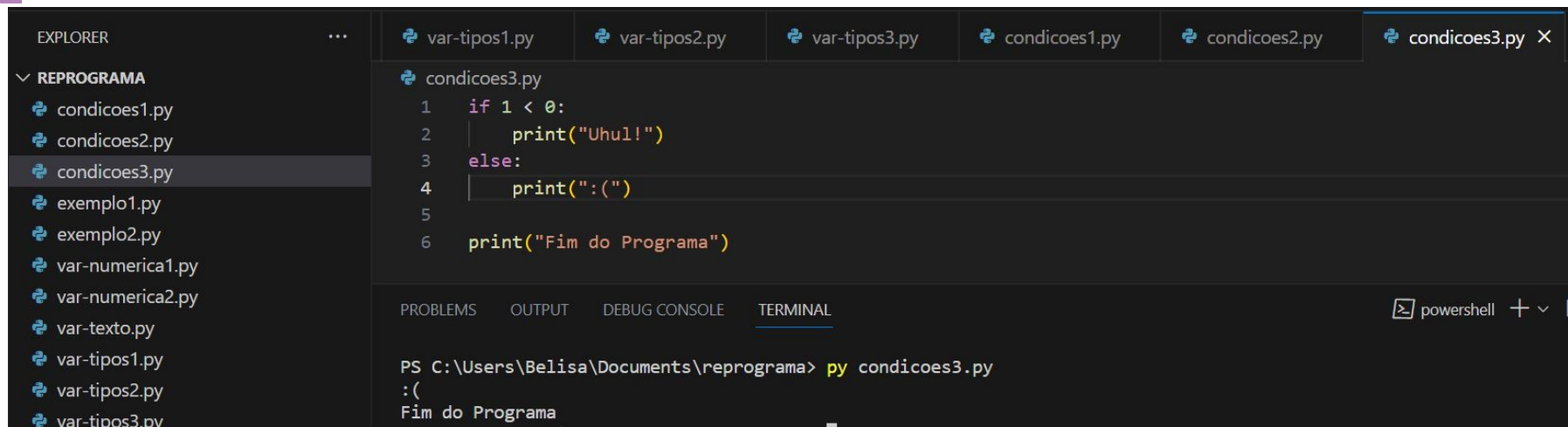
# Ferramentas de controle de fluxo - Condicionais

- Agora vamos adicionar uma alternativa ao nosso `if`
- Usaremos a instrução `else`, que em português significa algo como “senão”
- Leia mais ou menos como:

Se 1 é **menor** que 0, imprime “Uhul!”

Senão imprime “:(”

# Ferramentas de controle de fluxo - Condicionais



The screenshot displays a code editor interface with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'REPROGRAMA' containing several Python files. The file 'condicoes3.py' is selected. The code editor shows the following Python code:

```
condicoes3.py
1  if 1 < 0:
2      print("Uhul!")
3  else:
4      print(":(")
5
6  print("Fim do Programa")
```

Below the code editor, there is a terminal window with the following output:

```
PS C:\Users\Belisa\Documents\reprograma> py condicoes3.py
:(
Fim do Programa
```

# Ferramentas de controle de fluxo - Condicionais

- Agora vamos adicionar uma condicional na nossa alternativa ao `if`
- Usaremos a instrução `elif`, que é a contração de `else if`, e em português significa algo como “senão, e se...”
- Leia mais ou menos como:

Se 1 é **menor** que 0, imprime “Uhul!”

Senão, e se 1 é **maior** que 0, imprime “Agora sim!”



# Ferramentas de controle de fluxo - Condicionais

The screenshot displays a code editor interface with a file explorer on the left and a terminal at the bottom. The file explorer shows a directory named 'REPROGRAMA' containing several Python files. The file 'condicoes4.py' is selected and open in the editor. The code in 'condicoes4.py' uses conditional logic to print 'Uhul!' if a variable is less than 0, 'Agora sim!' if it is greater than 0, and 'Fim do Programa' at the end. The terminal at the bottom shows the command 'py condicoes4.py' being executed, which results in the output 'Agora sim!' followed by 'Fim do Programa'.

```
EXPLORER
```

- REPROGRAMA
  - condicoes1.py
  - condicoes2.py
  - condicoes3.py
  - condicoes4.py
  - condicoes5.py
  - exemplo1.py
  - exemplo2.py
  - var-numerica1.py
  - var-numerica2.py
  - var-texto.py
  - var-tipos1.py
  - var-tipos2.py
  - var-tipos3.py

```
condicoes4.py
```

```
1  if 1 < 0:
2      print("Uhul!")
3  elif 1 > 0:
4      print("Agora sim!")
5
6  print("Fim do Programa")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
```

```
PS C:\Users\Belisa\Documents\reprograma> py condicoes4.py
Agora sim!
Fim do Programa
PS C:\Users\Belisa\Documents\reprograma>
```

# Ferramentas de controle de fluxo - Condicionais

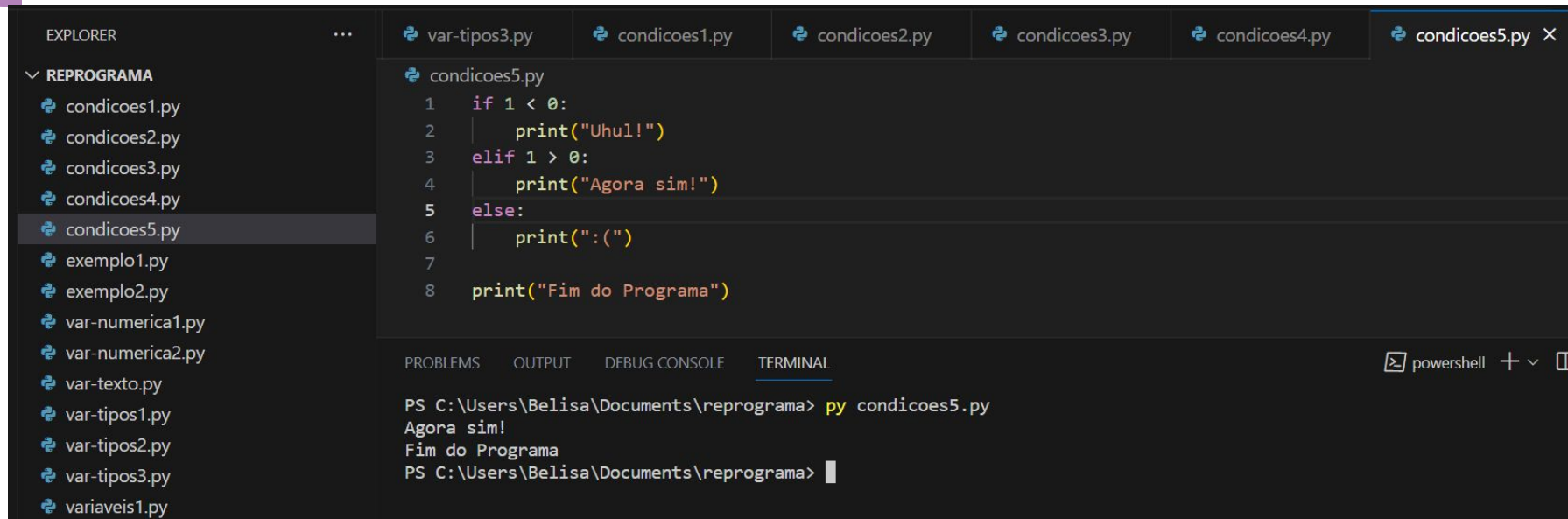
- Agora vamos juntas as três opções
- Leia mais ou menos como:

Se 1 é menor que 0, imprime "Uhul!"

Senão, e se 1 é **igual** a 0, imprime "Agora sim!"

Senão imprime ":("

# Ferramentas de controle de fluxo - Condicionais



The image shows a code editor interface with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'REPROGRAMA' containing several Python files. The file 'condicoes5.py' is selected. The code editor displays the contents of 'condicoes5.py', which is a Python script using conditional logic to print different messages based on the value of a variable '1'. The script is as follows:

```
1 if 1 < 0:
2     print("Uhul!")
3 elif 1 > 0:
4     print("Agora sim!")
5 else:
6     print(":(")
7
8 print("Fim do Programa")
```

Below the code editor, there is a terminal window showing the execution of the script. The terminal output is as follows:

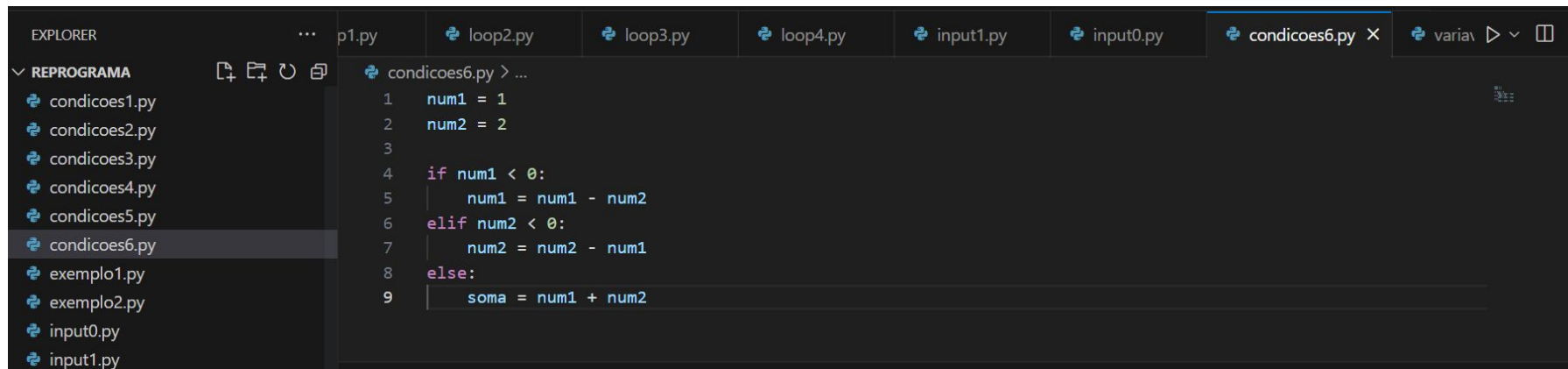
```
PS C:\Users\Belisa\Documents\reprograma> py condicoes5.py
Agora sim!
Fim do Programa
PS C:\Users\Belisa\Documents\reprograma>
```

# Ferramentas de controle de fluxo - Mão na massa

- Vamos começar com duas variáveis numéricas: numero1 e numero2
- Crie um programa que:
  - Subtraia num2 de num1 se num1 for negativo
  - Subtraia num1 de num2 se num1 Não for negativo e num2 for negativo
  - Se nenhum número for negativo, some as variáveis



# Ferramentas de controle de fluxo - Mão na massa



The image shows a code editor interface with a dark theme. On the left, the 'EXPLORER' sidebar displays a file tree under the folder 'REPROGRAMA'. The files listed are condicoes1.py, condicoes2.py, condicoes3.py, condicoes4.py, condicoes5.py, condicoes6.py (which is selected and highlighted), exemplo1.py, exemplo2.py, input0.py, and input1.py. The main editor area shows the content of 'condicoes6.py'. The code is as follows:

```
condicoes6.py > ...
1  num1 = 1
2  num2 = 2
3
4  if num1 < 0:
5      num1 = num1 - num2
6  elif num2 < 0:
7      num2 = num2 - num1
8  else:
9      soma = num1 + num2
```

# Ferramentas de controle de fluxo - Desafio

- Usando o programa do exercício anterior, acrescente a última regra
- Vamos começar com duas variáveis numéricas: numero1 e numero2
- Crie um programa que:
  - Subtraia num2 de num1 se num1 for negativo
  - Subtraia num1 de num2 se num2 for negativo
  - Se nenhum número for negativo, some as variáveis
  - Divida num1 por num2 se AMBOS forem negativos

```
EXPLORER
... op2.py loop3.py loop4.py input1.py input0.py condicoes6.py condicoes7.py X
▼ REPROGRAMA
  condicoes1.py
  condicoes2.py
  condicoes3.py
  condicoes4.py
  condicoes5.py
  condicoes6.py
  condicoes7.py
  exemplo1.py
  exemplo2.py
  input0.py
  input1.py

condicoes7.py > ...
1  num1 = 1
2  num2 = 2
3
4  if num1 < 0:
5      num1 = num1 - num2
6  elif num2 < 0:
7      num2 = num2 - num1
8  elif num1 > 0 and num2 > 0:
9      soma = num1 + num2
10 elif num1 < 0 and num2 < 0:
11     divisao = num1/num2
```

5 Minutinhos pra resfriar o cérebro  
Tomem água!





# Ferramentas de controle de fluxo - Laços de Repetição

- Agora que nós sabemos controlar condições de execução, podemos controlar repetições
- Vamos começar com o laço `while`, que em português significa enquanto
- Tal qual o `if`, a expressão `while` avalia se uma condição é verdadeira
- Leia como:

Enquanto a condição for verdadeira, imprime "Uhul!"

# Ferramentas de controle de fluxo - Loops

- Aqui nós precisamos ter cuidado! Um laço de repetição cuja *condição de parada* nunca será verdadeira vai criar um loop infinito. Isso é, nosso programa nunca vai sair do bloco interno de execução.
- Vamos mudar nossa condição para usar uma variável na comparação:

Enquanto num1 é menor que 0, imprime "Uhul!"

- E agora vamos trabalhar essa variável para garantir que nosso loop seja encerrado em algum ponto

# Ferramentas de controle de fluxo - Loops

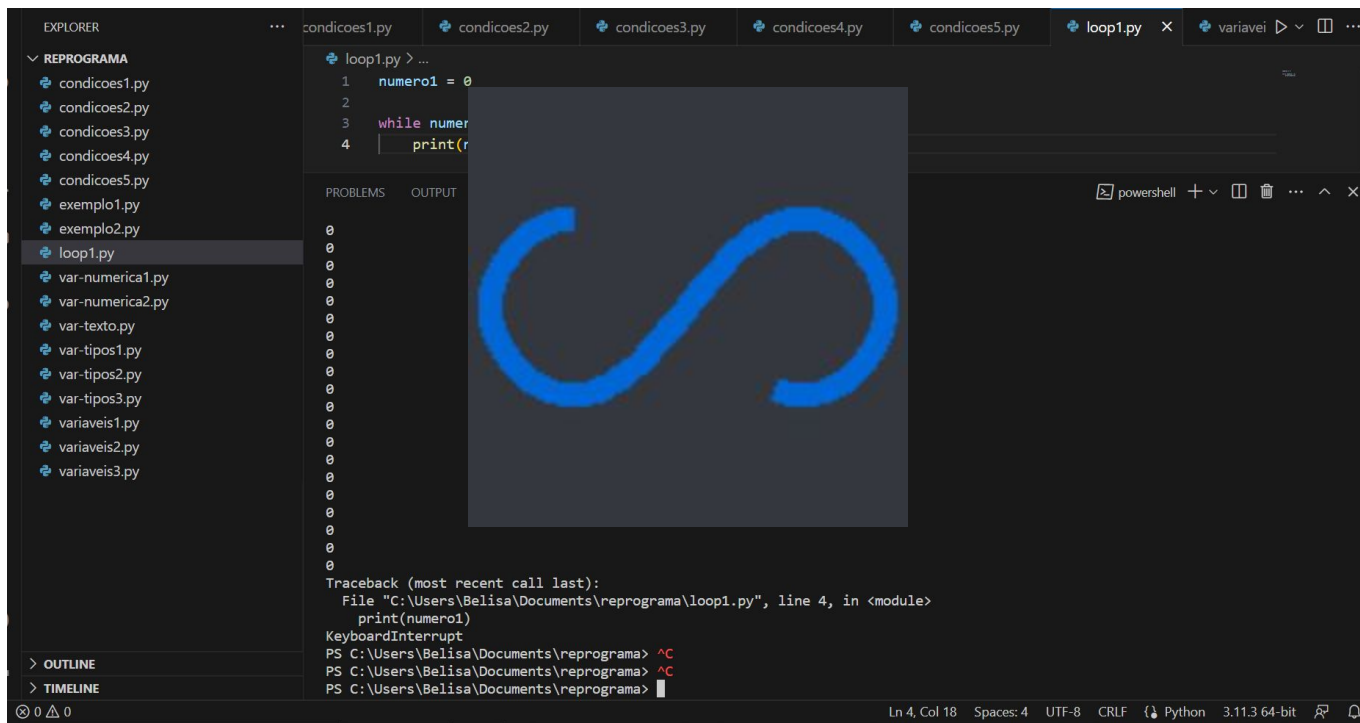
- Vamos criar uma laço que imprime os números de 1 a 5:

Enquanto numero1 é menor que 5, imprime numero1

- Nós precisamos iniciar a variável `num1` antes do laço, senão o python não saberá seu valor:

```
1 num1 = 0
```

# Ferramentas de controle de fluxo - Loops



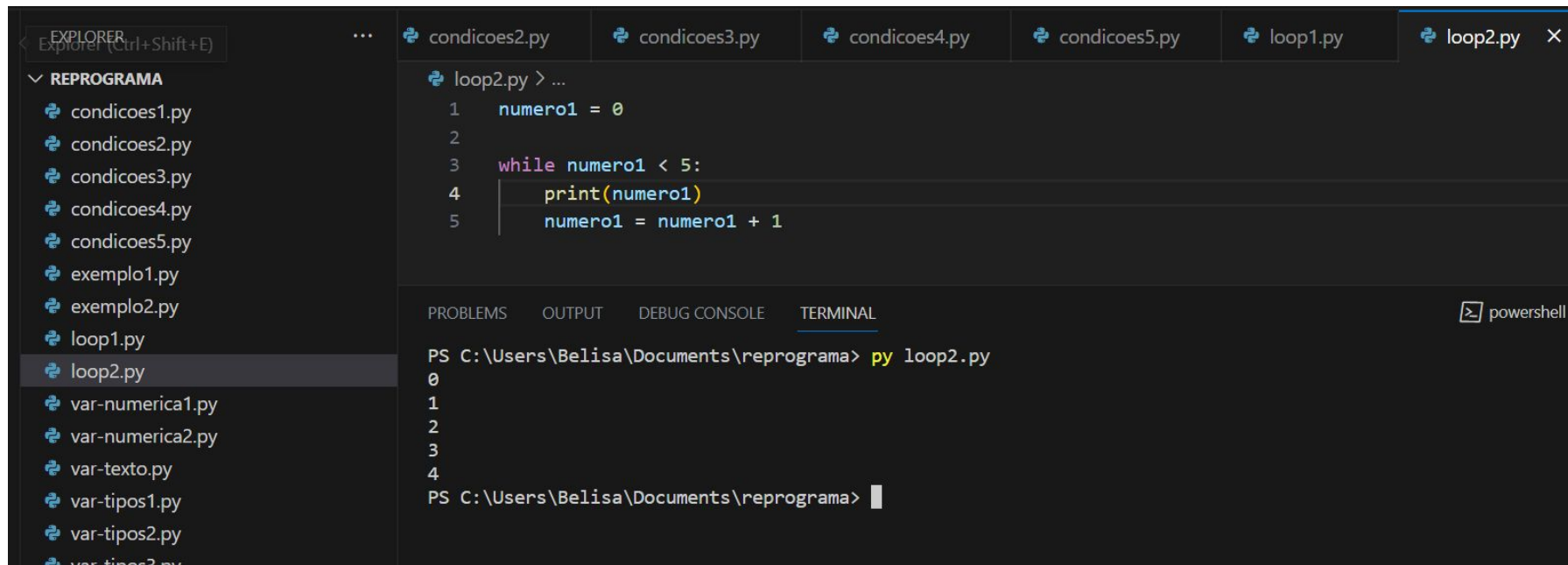
The screenshot shows a Python IDE with a dark theme. The Explorer panel on the left lists files in a folder named 'REPROGRAMA', including 'condicoes1.py' through 'condicoes5.py', 'exemplo1.py', 'exemplo2.py', 'loop1.py', and several 'var-' and 'variaveis-' files. 'loop1.py' is selected. The main editor shows the code for 'loop1.py':

```
1 numero1 = 0
2
3 while numero1 < 10:
4     print(numero1)
```

The OUTPUT panel shows the program's execution, printing the numbers 0 through 9 on separate lines. A large, semi-transparent infinity symbol is overlaid on the output. Below the output, a traceback message indicates a 'KeyboardInterrupt' occurred at line 4. The status bar at the bottom shows 'Ln 4, Col 18', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', and '3.11.3 64-bit'.

# Ferramentas de controle de fluxo - Loops

- Agora vamos incrementar a variável num1 a cada execução do loop



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'REPROGRAMA' containing several Python files. The file 'loop2.py' is selected. The code editor shows the following Python code:

```
1 numero1 = 0
2
3 while numero1 < 5:
4     print(numero1)
5     numero1 = numero1 + 1
```

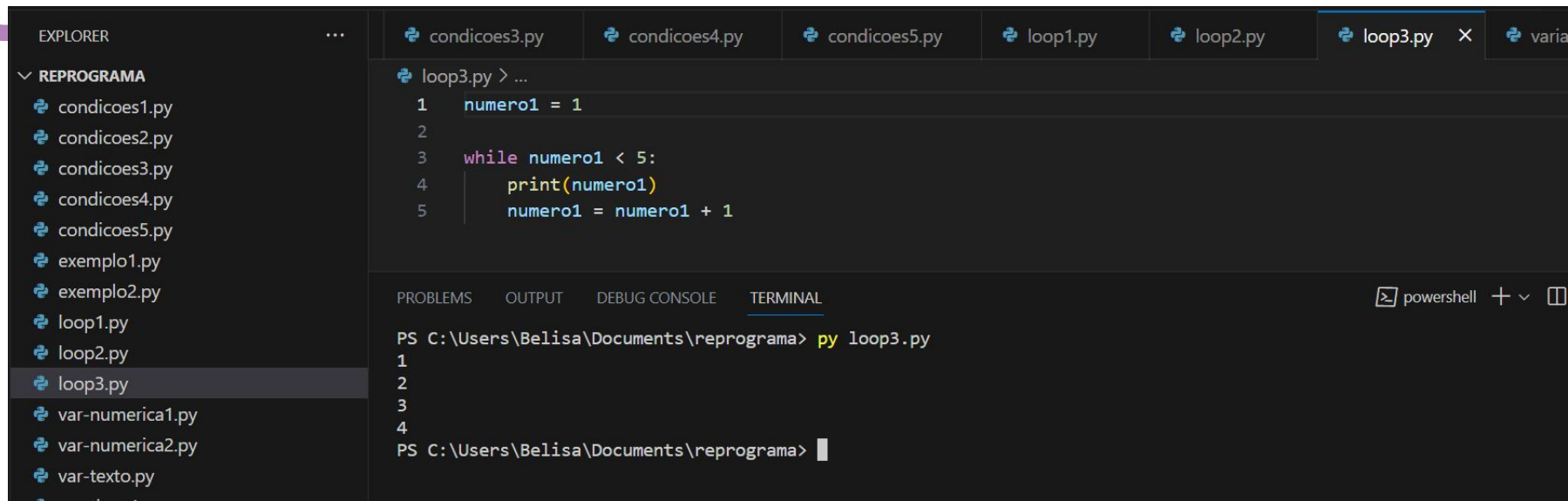
Below the code editor, there is a terminal window showing the output of the program. The terminal prompt is 'PS C:\Users\Belisa\Documents\reprograma>'. The command 'py loop2.py' has been executed, and the output is:

```
0
1
2
3
4
PS C:\Users\Belisa\Documents\reprograma>
```

# Ferramentas de controle de fluxo - Loops - Perguntas

- Quantas vezes o loop foi executado?
- Nós conseguimos imprimir os números de 1 a 5?
- Se não conseguimos, porque não?

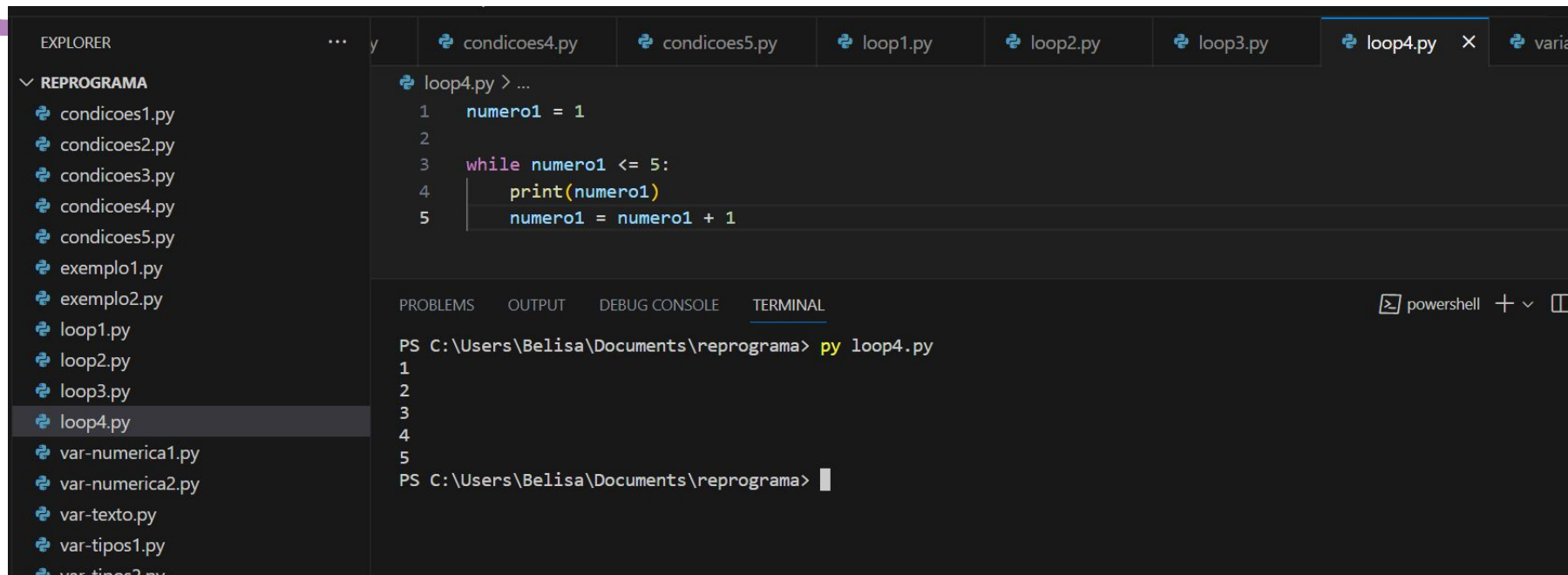
# Ferramentas de controle de fluxo - Loops



The screenshot displays a code editor interface with a file explorer on the left and a terminal at the bottom. The file explorer shows a directory named 'REPROGRAMA' containing several Python files, with 'loop3.py' selected. The editor window shows the code for 'loop3.py', which is a while loop that prints the value of 'numero1' and increments it by 1, repeating this process five times. The terminal at the bottom shows the command 'py loop3.py' being executed, resulting in the output '1', '2', '3', and '4'.

```
EXPLORER  
▼ REPROGRAMA  
  condicoes1.py  
  condicoes2.py  
  condicoes3.py  
  condicoes4.py  
  condicoes5.py  
  exemplo1.py  
  exemplo2.py  
  loop1.py  
  loop2.py  
  loop3.py  
  var-numerica1.py  
  var-numerica2.py  
  var-texto.py  
  ...  
condicoes3.py condicoes4.py condicoes5.py loop1.py loop2.py loop3.py varia  
loop3.py > ...  
1  numero1 = 1  
2  
3  while numero1 < 5:  
4      print(numero1)  
5      numero1 = numero1 + 1  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL  
PS C:\Users\Belisa\Documents\reprograma> py loop3.py  
1  
2  
3  
4  
PS C:\Users\Belisa\Documents\reprograma> |
```

# Ferramentas de controle de fluxo - Loops



The screenshot displays a code editor interface with a file explorer on the left and a terminal at the bottom. The file explorer shows a directory named 'REPROGRAMA' containing several Python files, with 'loop4.py' selected. The editor window shows the code for 'loop4.py', which is a while loop that prints the value of 'numero1' from 1 to 5. The terminal shows the command 'py loop4.py' being executed, resulting in the output '1', '2', '3', '4', and '5' on separate lines.

```
EXPLORER
```

- REPROGRAMA
  - condicoes1.py
  - condicoes2.py
  - condicoes3.py
  - condicoes4.py
  - condicoes5.py
  - exemplo1.py
  - exemplo2.py
  - loop1.py
  - loop2.py
  - loop3.py
  - loop4.py
  - var-numerica1.py
  - var-numerica2.py
  - var-texto.py
  - var-tipos1.py
  - var-tipos2.py

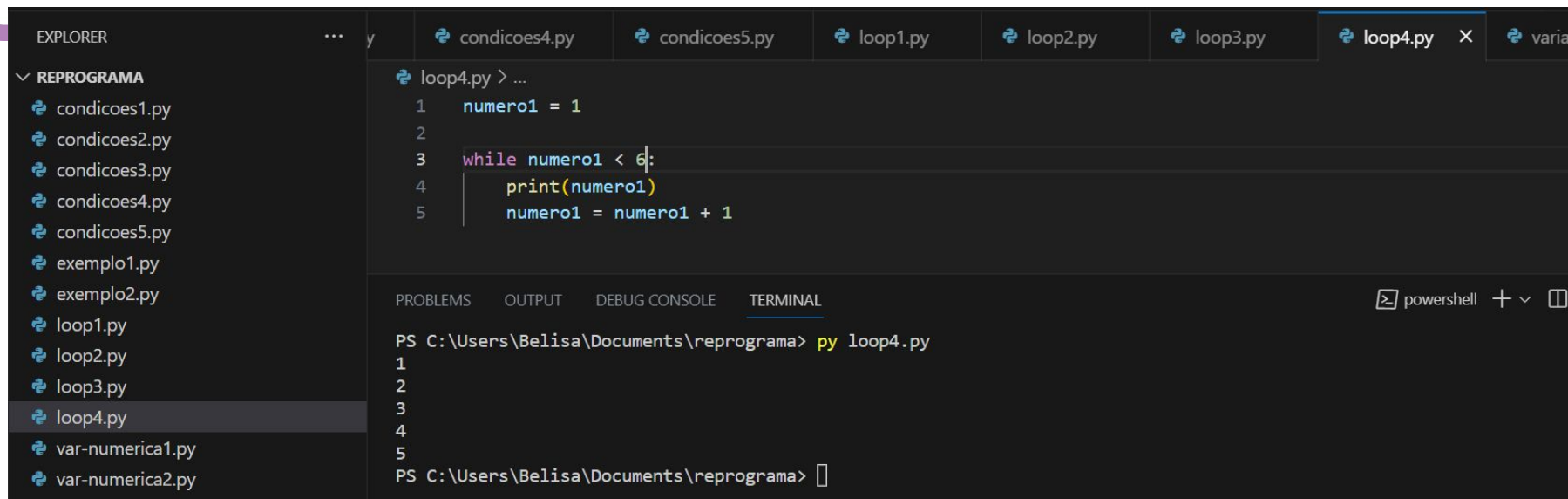
```
loop4.py > ...
1  numero1 = 1
2
3  while numero1 <= 5:
4      print(numero1)
5      numero1 = numero1 + 1
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
```

```
PS C:\Users\Belisa\Documents\reprograma> py loop4.py
1
2
3
4
5
PS C:\Users\Belisa\Documents\reprograma>
```



# Ferramentas de controle de fluxo - Loops



The image shows a code editor interface with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'REPROGRAMA' with several Python files. The code editor shows a Python script named 'loop4.py' with a while loop. Below the code editor is a terminal window showing the execution of the script.

**EXPLORER**

- REPROGRAMA
  - condicoes1.py
  - condicoes2.py
  - condicoes3.py
  - condicoes4.py
  - condicoes5.py
  - exemplo1.py
  - exemplo2.py
  - loop1.py
  - loop2.py
  - loop3.py
  - loop4.py
  - var-numerica1.py
  - var-numerica2.py

**loop4.py**

```
1 numero1 = 1
2
3 while numero1 < 6:
4     print(numero1)
5     numero1 = numero1 + 1
```

**TERMINAL**

```
PS C:\Users\Belisa\Documents\reprograma> py loop4.py
1
2
3
4
5
PS C:\Users\Belisa\Documents\reprograma>
```

# Rodada Bônus



# Rodada Bônus



*Seja gentil  
consigo mesmo!*

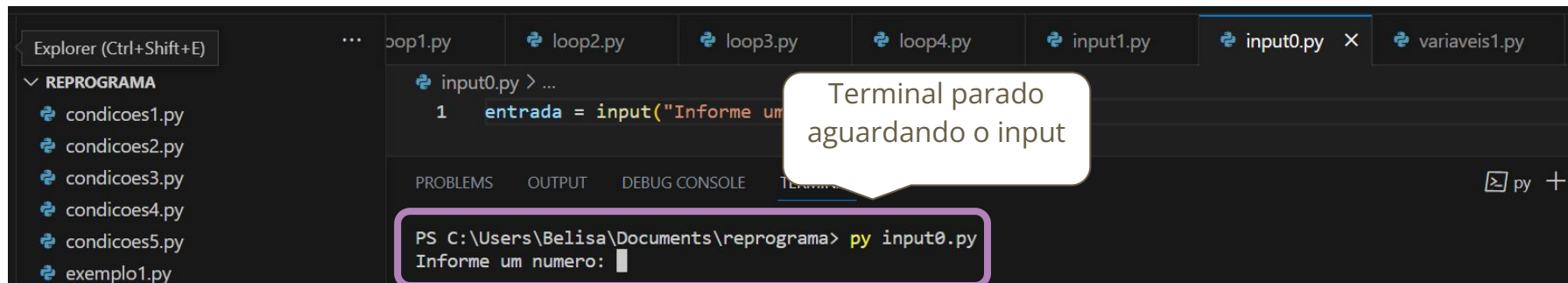
## Recebendo dados durante a execução

- Para nossos exemplos ficarem um pouco mais legais, vamos receber inputs (entradas) e trabalhar com eles. Python permite fazer isso com uma função bem simples:

```
entrada = input()
```

# Recebendo dados durante a execução

- Quando o código executar a função `input`, a execução é parada até que um valor seja informado no próprio terminal.
- Para ficar mais fácil saber o que queremos que o usuário informe, podemos adicionar texto na função, que será impressa no momento em que o usuário deve informar o valor:



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a folder named 'REPROGRAMA' containing several Python files: condicoes1.py, condicoes2.py, condicoes3.py, condicoes4.py, condicoes5.py, and exemplo1.py. The main editor area has several tabs open: loop1.py, loop2.py, loop3.py, loop4.py, input1.py, input0.py (selected), and variaveis1.py. The selected tab 'input0.py' shows the following code:

```
input0.py > ...  
1 entrada = input("Informe um numero: ")
```

Below the code editor, the TERMINAL pane is active, showing the command prompt output:

```
PS C:\Users\Belisa\Documents\reprograma> py input0.py  
Informe um numero: 
```

A callout bubble points to the terminal, stating: "Terminal parado aguardando o input".

# Recebendo dados durante a execução - Mão na massa

- Desafio:
  - Criar duas variáveis a partir de input
  - Concatenar as duas variáveis em uma terceira variável chamada concat
  - Imprimir a variável concat
  - Descobrir o tipo da variável concat





The image shows a Visual Studio Code editor window with a dark theme. On the left, the 'EXPLORER' sidebar displays a file tree for a project named 'REPROGRAMA'. The files listed are: `condicoes1.py`, `condicoes2.py`, `condicoes3.py`, `condicoes4.py`, `condicoes5.py`, `exemplo1.py`, `exemplo2.py`, `input1.py` (selected), `loop1.py`, `loop2.py`, `loop3.py`, `loop4.py`, `var-numerica1.py`, `var-numerica2.py`, `var-texto.py`, and `var-tipos1.py`. The top of the editor shows several open tabs: `loop1.py`, `loop2.py`, `loop3.py`, `loop4.py`, `input1.py` (active), `variaveis1.py`, and `variaveis2.py`. The main editor area displays the code for `input1.py`:

```
1 entrada1 = input("Informe o primeiro valor: ")
2 entrada2 = input("Informe o segundo valor: ")
3
4 print(type(entrada1))
5 print(type(entrada2))
6
7 concat = entrada1 + entrada2
8 print("Concat: " + concat)
9
10 print("Tipo da variável concat: ")
```

A yellow callout bubble with the text 'Terminal parado aguardando o input' points to the terminal area. The bottom of the editor features a panel with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active, showing the command prompt output:

```
PS C:\Users\Belisa\Documents\reprograma> py input1.py
Informe o primeiro valor: 
```

The terminal input line is highlighted with a purple rounded rectangle.

EXPLORER

REPROGRAMA

condicoes1.py

condicoes2.py

condicoes3.py

condicoes4.py

condicoes5.py

exemplo1.py

exemplo2.py

input1.py

loop1.py

loop2.py

loop3.py

loop4.py

var-numerica1.py

var-numerica2.py

var-texto.py

var-tipos1.py

var-tipos2.py

var-tipos3.py

variaveis1.py

loop1.py

loop2.py

loop3.py

loop4.py

input1.py

variaveis1.py

variaveis2.py

input1.py > ...

1 entrada1 = input("Informe o primeiro valor: ")

2 entrada2 = input("Informe o segundo valor: ")

3

4 print(type(entrada1))

5 print(type(entrada2))

6

7 concat = entrada1 + entrada2

8 print("Concat: " + concat)

9

10 print("Tipo da variável concat: " + str((type(concat))))

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

powershell

+

▼

PS C:\Users\Belisa\Documents\reprograma> py input1.py

Informe o primeiro valor: Reprograma

Informe o segundo valor: é \*\*\*\*

<class 'str'>

<class 'str'>

Concat: Reprograma é \*\*\*\*

Tipo da variável concat: <class 'str'>

PS C:\Users\Belisa\Documents\reprograma>



# Loops + Input - Mão na massa

- Vamos criar uma simulação de login
- Primeiro, vamos criar duas variáveis: usuario; e senha.



# Ferramentas de controle de fluxo - Loops - Mão na massa

- Agora, vamos pedir que o usuário informe seu usuário e senha:

```
1  usuario = "reprograma"
2  senha = "12345"
3
4  usuario_informado = input('Informe o usuário: ')
5  senha_informada = input('Informe a senha: ')
```

# Ferramentas de controle de fluxo - Loops - Mão na massa

- Vamos conferir se os dados que o usuário informou batem com o usuário e senha do programa:

```
1  usuario = "reprograma"
2  senha = "12345"
3
4  usuario_informado = input('Informe o usuário: ')
5  senha_informada = input('Informe a senha: ')
6
7  if usuario == usuario_informado and senha == senha_informada:
8      print('Login bem sucedido!')
9  else:
10     print('Nome de usuário ou senha incorreta')
11
```

Operador  
lógico, em  
português: E

# Ferramentas de controle de fluxo - Loops - Mão na massa

- Pronto, agora nós sabemos fazer o login, mas somente uma vez.
- O próximo passo agora é fazer com que o programa execute até que o login seja bem sucedido.
- Para isso, vamos pensar alguns detalhes:
  - Nós precisamos definir usuário e senha a cada execução do loop?
  - Nós precisamos que o usuário informe usuário e senha a cada loop?
  - Como nós vamos fazer a condição que encerra o loop ser verdadeira?

# Ferramentas de controle de fluxo - Loops - Mão na massa

- Nós precisamos definir usuário e senha a cada execução do loop?
  - Não! Nós não queremos que usuário e senha salvos mudem, então eles não precisam ser definidos dentro do loop.
- Nós precisamos que o usuário informe usuário e senha a cada loop?
  - Sim! Se o usuário não puder corrigir as informações informadas, como ele vai conseguir fazer login?
- Como nós vamos fazer a condição que encerra o loop ser verdadeira?
  - Vamos usar uma variável para isso.

```
1  usuario = "reprograma"
2  senha = "12345"
3
4  login_bem_sucedido = False
5
6  while not login_bem_sucedido:
7      usuario_informado = input('Informe o usuário: ')
8      senha_informada = input('Informe a senha: ')
9
10     if usuario == usuario_informado and senha == senha_informada:
11         login_bem_sucedido = True
12     else:
13         print('Usuário ou senha incorretos')
14         login_bem_sucedido = False
15
16  print('Login bem sucedido!')
17
```

Não  
"Enquanto o login  
não for bem  
sucedido"

Falso

False

Verdadeiro

True

# Desafio

- Nos próximos slides, nosso desafio de avaliação será descrito.
- Você deve escrever seu programa para satisfazer o que é pedido.
- No final, você deve enviar o arquivo através do typeform fornecido pela monitora.

# Desafio - Calculadora! - Versão 0

- Crie duas variáveis com números
- Escolha uma operação matemática para os números
  - Por exemplo:  $\text{num1} * \text{num2}$
- Imprima o resultado



# Desafio - Calculadora! - Versão 1

- Crie duas variáveis com números
- Crie uma regra para escolher a operação
  - Por exemplo, se o primeiro valor for positivo, some. Senão, subtraia.
  - Use `if`, `elif` e `else` para escolher a operação
- Imprima o resultado

## Desafio - Calculadora! - Versão 2

- Crie duas variáveis com números
- Crie uma regra para escolher a operação
  - Por exemplo, se o primeiro valor for positivo, some. Senão, subtraia.
  - Use `if`, `elif` e `else` para escolher a operação
- Imprima o resultado
- Faça com que o programa escolha a operação e calcule 4 vezes
  - Use um laço `while` para isso

# Desafio Bônus



*Seja gentil  
consigo mesmo!*

## Desafio - Calculadora! - V3

- Crie duas variáveis com números
- Use um input para receber a operação (+, -, \* ou /)
- Execute a operação e imprima o valor
  - Use `if`, `elif` e `else` para descobrir a operação informada
- Faça com que o programa continue rodando (pedindo input e imprimindo o resultado) até que o usuário digite uma operação inválida

# Desafio - Calculadora! - V4

- Crie duas variáveis usando inputs
  - Use a função `int()` para converter os inputs em números
  - Informe ao usuário que se o valor digitado não for numérico, o programa será encerrado com erro
- Use um input para receber a operação (+, -, \* ou /)
- Execute a operação e imprima o valor
  - Use `if`, `elif` e `else` para descobrir a operação informada
- Faça com que o programa continue rodando (pedindo input e imprimindo o resultado) até que o usuário digite uma operação inválida

# Desafio Bônus



# Desafio - Calculadora! - V5

- Crie as duas variáveis usando inputs e replique os demais passos da v4
  - Use a função `int()` para converter os inputs em números
  - Não permita que o usuário informe um valor que não é numérico (trate o erro)

**Atenção:** essa versão é um **desafio**. Nós não aprendemos como tratar o erro durante a oficina. Busque na internet como fazer o tratamento. Se não conseguir, não se sinta mal. Isso é um desafio, não uma obrigação!

Nós abordamos MUITO conhecimento hoje. Seja gentil com você mesma!



Uhhhhhhhhhhhh!





