



---

---

# Python para análise de dados

— Semana 3 —

---

---





# Cronograma estimado



09h00 - 12h00

13h30 - 15h30

15h30 - 16h15

16h15 - 17h00

Condicionais e Operadores  
Lógicos

Erros e seus tipos

Usando o Debug

Try e Except

# Recapitulando a aula 2

- Valores, variáveis e tipos de valores
- Inputs
- Operações
- Formatação de strings
- Indentação
- Funções

# 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, que será identificado pela indentação, tal qual ocorre em funções
- Depois de executar (ou não!) o bloco interno, a execução segue 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!"


# Operadores de comparação

## Operadores Relacionais

Porque não  
usar somente  
um "="?

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

# 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 directory named 'REPROGRAMA' containing several Python files. The active file is 'condicoes1.py'. The code in the editor is as follows:

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

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

# Condicionais

- Vamos tornar nossos exemplos mais próximos do mundo real usando variáveis na comparação.
- Para os próximos exemplos, imaginem que estamos decidindo se um aluno será aprovado ou não em uma disciplina.
- Para isso, usaremos a variável nota e com ela decidiremos qual ação tomar.



```
1  nota = 7
2
3  if nota >= 7:
4      print("Aprovado!")
5
6  print("Fim do Programa")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

bash - condicionais

```
• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicoes1-var.py
Aprovado!
Fim do Programa
○ belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$
```

## Condicionais - Operadores lógicos

- É possível fazer mais de uma comparação no mesmo If. Essas comparações podem ser do tipo “ou” ou do tipo “e”.
- Para escrever o operador lógico e no código, usaremos a palavra reservada **and**.
- Seguindo o exemplo anterior, além da nota, o aluno também deverá ter presença superior a 70% para ser aprovado.
- Vamos começar pela e. Lemos ela como:

Se a **condição1** é **verdadeira** **E** a **condição2** é **verdadeira**, imprime “Uhul!”

Se nota é **maior ou igual** a 7 e presença é **maior** a 75  
imprime “Aprovado”

# Condicionais

aula3 > condicionais >  condicoes-and.py > ...

```
1  nota = 7
2  presenca = 80
3
4  if nota >= 7 and presenca > 75:
5      print("Aprovado!")
6
7  print("Fim do Programa")
```


PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

GITLENS

 bash - condicionais

```
• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicoes-and.py
Aprovado!
Fim do Programa
```

# Condicionais - Operadores lógicos

- Outra comparação possível é usando “ou”.
- Para escrever o operador lógico e no código, usaremos a palavra reservada **or**.
- Seguindo o exemplo anterior, o professor decidiu que basta ao aluno ter nota acima de 7 OU ter presença superior a 70% para ser aprovado.
- Vamos começar pela e. Lemos ela como:

Se a **condição1** é **verdadeira** **OU** a **condição2** é **verdadeira**, imprime “Uhul!”

Se nota é **maior ou igual** a 7 ou presença é **maior** a 75

imprime “Aprovado”

# Condicionais

aula3 > condicionais >  condicoes-or.py > ...

```
1  nota = 7
2  presenca = 20
3
4  if nota >= 7 or presenca > 75:
5      print("Aprovado!")
6
7  print("Fim do Programa")
```


PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

GITLENS

 bash - condicionais +

```
• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicoes-or.py
Aprovado!
Fim do Programa
```

# 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 a condição é verdadeira imprime “Aprovado!”

Senão imprime “Reprovado”

# Condicionais

```
1  nota = 6
2  presenca = 60
3
4  if nota >= 7 or presenca == 100:
5      print("Aprovado!")
6  elif nota < 7 and nota > 5:
7      if presenca >= 90:
8          print("Aprovado!")
9      elif presenca > 50:
10         print("Apto a fazer recuperação")
11     else:
12         print("Reprovado")
13 else:
14     print("Reprovado")
15
16 print("Fim do Programa")
```

PROBLEMS 1

OUTPUT

DEBUG CONSOLE

TERMINAL

GIT LENS

bash - condicionais +

```
• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicoes-aninhadas.py
Apto a fazer recuperação
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...”
- A regra do professor mudou: agora se um aluno tiver nota insuficiente, mas tiver 100% de presença, ele está aprovado mesmo assim.
- Leia mais ou menos como:

Se a `condição1` é verdadeira imprime “Aprovado!”

Senão e se a `condição2` é verdadeira imprime “Aprovado por presença”



# Ferramentas de controle de fluxo - Condicionais

```
aula3 > condicionais > condicoes-elif.py > ...
1  nota = 6
2  presenca = 100
3
4  if nota >= 7:
5      print("Aprovado!")
6  elif presenca == 100:
7      print("Aprovado por presença")
8
9  print("Fim do Programa")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

bash - condicionais +

```
● belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicoes-elif.py
Aprovado por presença
Fim do Programa
```

# Ferramentas de controle de fluxo - Condicionais

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

Se a condição1 é verdadeira imprime “Aprovado!”

Senão e se a condição2 é verdadeira imprime “Aprovado por presença”


Senão imprime “Reprovado”

# Ferramentas de controle de fluxo - Condicionais

aula3 > condicionais >  condicoes-elif-else.py > ...

```
1  nota = 6
2  presenca = 100
3
4  if nota >= 7:
5      print("Aprovado!")
6  elif presenca == 100:
7      print("Aprovado por presença")
8  else:
9      print("Reprovado")
10
11 print("Fim do Programa")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

 bash - condicionais + ▾

```
• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicoes-elif-else.py
Aprovado por presença
Fim do Programa
```

# Condicionais aninhadas

- Por vezes, é necessário fazer comparações dentro de outras.
- Vamos supor que o professor agora criou uma regra super complicada para aprovação por presença.
- Se o aluno atingiu a média ou 100% da presença, está aprovado.
- Se o aluno não atingiu a média para aprovação, mas tiver nota acima de 5 e 90% de presença, está aprovado.
- Se o aluno tiver nota acima de 6, e tiver presença acima de 50%, ele tem direito a recuperação.
- Se não tiver nem nota acima de 6, nem presença acima de 50%, o aluno está reprovado.

# Condicionais aninhadas

aula3 > condicionais > condicionais-repeticao.py > ...

```
1  nota = 6
2  presenca = 60
3
4  if nota >= 7 or presenca == 100:
5      print("Aprovado!")
6  elif (nota < 7 and nota > 5) and presenca >= 90:
7      print("Aprovado!")
8  elif (nota < 7 and nota > 5) and presenca > 50:
9      print("Apto a fazer recuperação")
10 else:
11     print("Reprovado")
12
13 print("Fim do Programa")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

bash - condicionais + ▾

```
• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicionais-repeticao.py
Apto a fazer recuperação
Fim do Programa
```

# Condicionais aninhadas

- Essas comparações podem todas ser feitas usando somente if, elif e else e os operadores and e or.
- Mas também é possível evitar conferir várias vezes a mesma condição usando if aninhados. Isso é, usar uma cláusula if dentro de outra cláusula if (ou elif, ou else).

aula3 > condicionais > condicoes-aninhadas.py > ...

```
1  nota = 6
2  presenca = 60
3
4  if nota >= 7 or presenca == 100:
5      print("Aprovado!")
6  elif nota < 7 and nota > 5:
7      if presenca >= 90:
8          print("Aprovado!")
9      elif presenca > 50:
10         print("Apto a fazer recuperação")
11 else:
12     print("Reprovado")
13
14 print("Fim do Programa")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

bash - condicionais +

```
• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/condicionais$ python3 condicoes-aninhadas.py
Apto a fazer recuperação
Fim do Programa
```

# 5 Minutinhos de alegria - Tomem água!



Toma awa



# Identificando e corrigindo erros

- Um passo importante para programar melhor e com mais eficiência é saber identificar e corrigir erros dos programas.
- Erros irão acontecer. Eu garanto.
- Indentação errada, confundir variáveis, esquecer de fechar um parênteses...
- Saber identificar e entender as mensagens de erro certamente ajudará MUITO!
- Vamos primeiro separar alguns tipos de erros antes de roda e depois entenderemos como acompanhar passo a passo no programa.

# Erros de sintaxe

- O primeiro tipo de erro, e provavelmente o mais comum, é o erro de sintaxe.
- Ele ocorre quando escrevemos algo no código que não pode ser interpretado corretamente.
- Por exemplo, esquecer os dois pontos após um condicional
- Ou esquecer o recuo após a cláusula.
- Vamos criar um erro de propósito no nosso último exemplo e analisar a resposta. Como o vscode identifica o erro antes do código ser executado, usaremos o google colab nos exemplos.

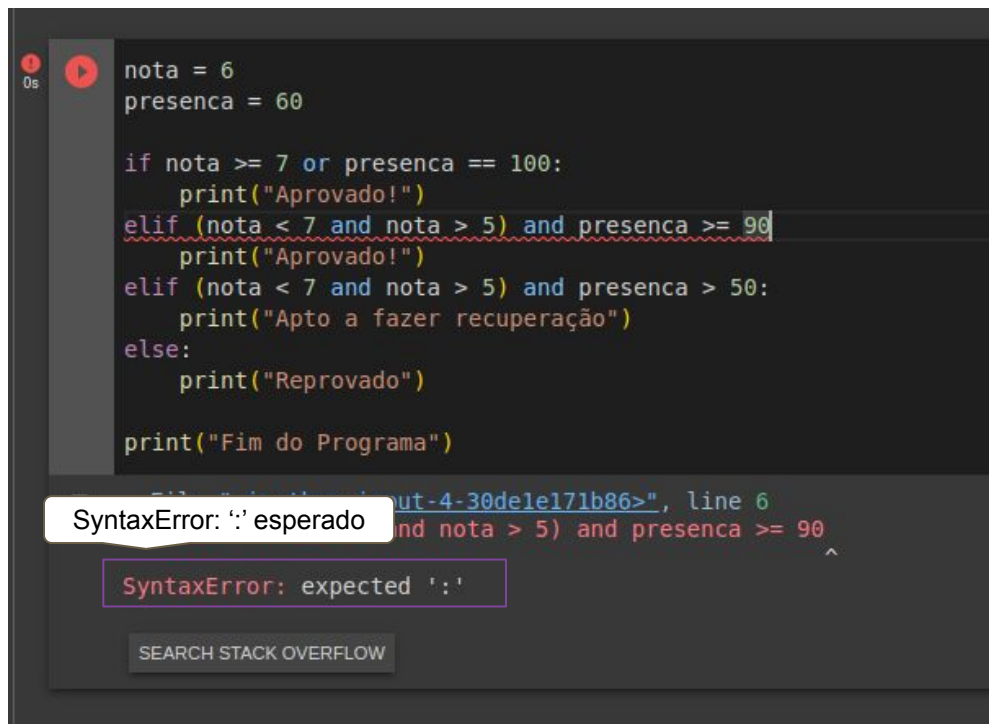
# Erros de sintaxe - Jogo dos 1 erros

```
▶ nota = 6
  presença = 60

  if nota >= 7 or presença == 100:
      print("Aprovado!")
  elif (nota < 7 and nota > 5) and presença >= 90
      print("Aprovado!")
  elif (nota < 7 and nota > 5) and presença > 50:
      print("Apto a fazer recuperação")
  else:
      print("Reprovado")

  print("Fim do Programa")
```

# Erros de sintaxe - Jogo dos 1 erros



```
0s [play icon]
nota = 6
presenca = 60

if nota >= 7 or presenca == 100:
    print("Aprovado!")
elif (nota < 7 and nota > 5) and presenca >= 90
    print("Aprovado!")
elif (nota < 7 and nota > 5) and presenca > 50:
    print("Apto a fazer recuperação")
else:
    print("Reprovado")

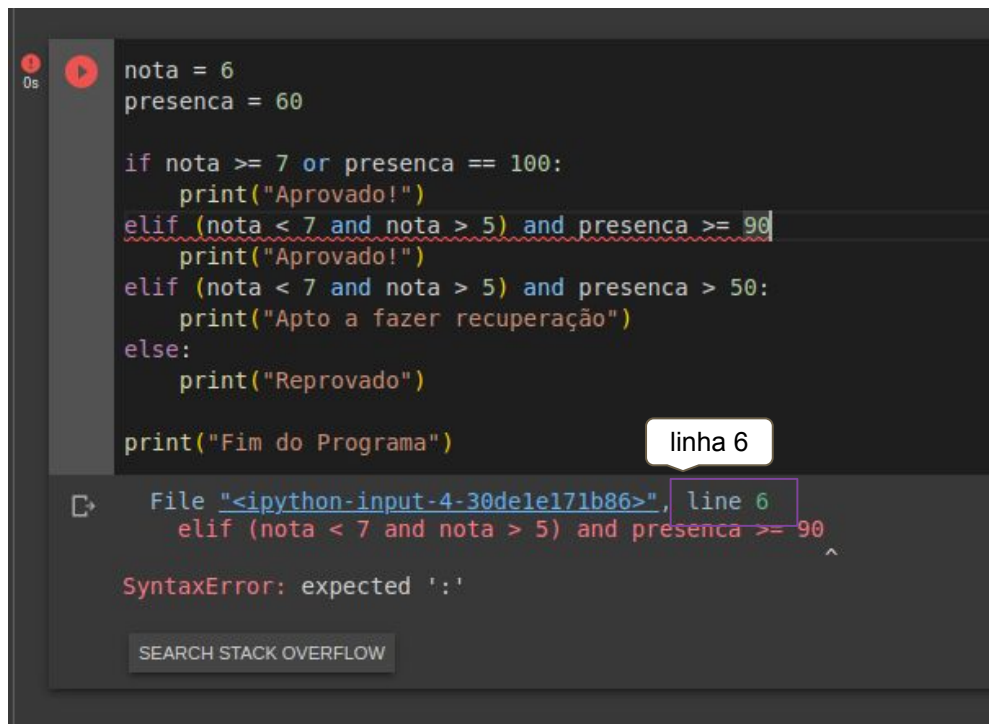
print("Fim do Programa")
```

SyntaxError: ':' esperado

SyntaxError: expected ':'

SEARCH STACK OVERFLOW

# Erros de sintaxe - Jogo dos 1 erros



```
0s [play icon] nota = 6
presenca = 60

if nota >= 7 or presenca == 100:
    print("Aprovado!")
elif (nota < 7 and nota > 5) and presenca >= 90
    print("Aprovado!")
elif (nota < 7 and nota > 5) and presenca > 50:
    print("Apto a fazer recuperação")
else:
    print("Reprovado")

print("Fim do Programa")
```

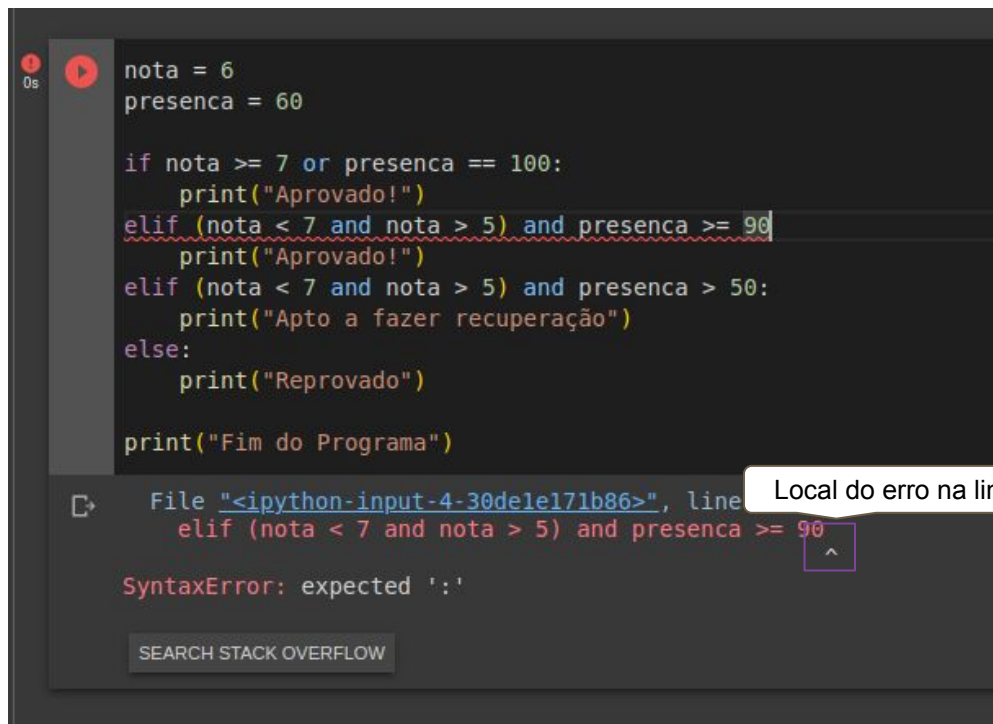
linha 6

File "<ipython-input-4-30de1e171b86>", line 6  
elif (nota < 7 and nota > 5) and presenca >= 90  
^

SyntaxError: expected ':'

SEARCH STACK OVERFLOW

# Erros de sintaxe - Jogo dos 1 erros



```
0s [play icon] nota = 6
presenca = 60

if nota >= 7 or presenca == 100:
    print("Aprovado!")
elif (nota < 7 and nota > 5) and presenca >= 90
    print("Aprovado!")
elif (nota < 7 and nota > 5) and presenca > 50:
    print("Apto a fazer recuperação")
else:
    print("Reprovado")

print("Fim do Programa")
```

File "<ipython-input-4-30de1e171b86>", line 10  
elif (nota < 7 and nota > 5) and presenca >= 90  
SyntaxError: expected ':'

Local do erro na linha

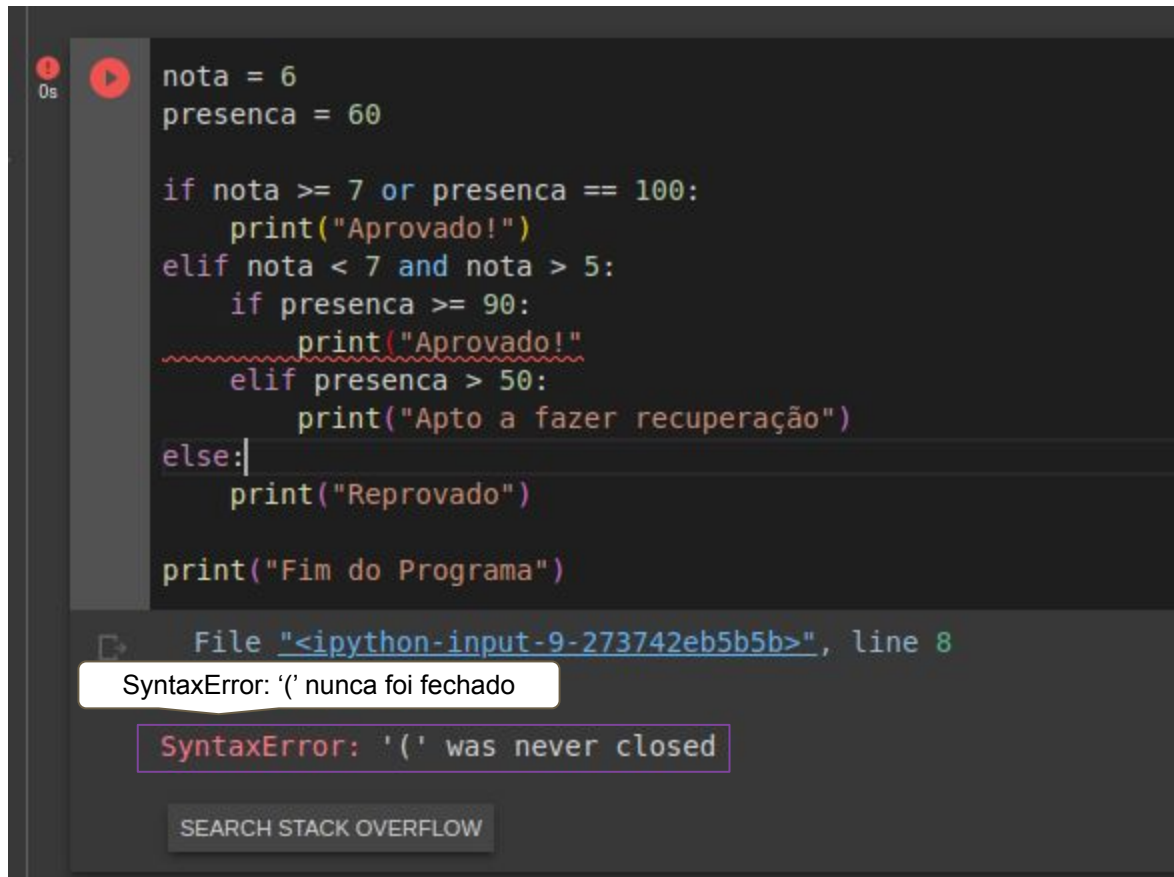
SEARCH STACK OVERFLOW

# Erros de sintaxe - Jogo dos 1 erros

```
▶ nota = 6
  presença = 60

  if nota >= 7 or presença == 100:
    print("Aprovado!")
  elif nota < 7 and nota > 5:
    if presença >= 90:
      print("Aprovado!")
    elif presença > 50:
      print("Apto a fazer recuperação")
  else:
    print("Reprovado")

  print("Fim do Programa")
```



The image shows a Python IDE interface with a dark theme. On the left, there is a sidebar with a red error icon and a play button icon. The main area displays a Python script. The script defines variables 'nota' and 'presenca', then uses a series of conditional statements to check for approval based on grade and attendance. A syntax error is highlighted on line 8, where a parenthesis is not closed. Below the code, the error message is displayed in a white box, and a search bar for Stack Overflow is visible at the bottom.

```
nota = 6
presenca = 60

if nota >= 7 or presenca == 100:
    print("Aprovado!")
elif nota < 7 and nota > 5:
    if presenca >= 90:
        print("Aprovado!")
    elif presenca > 50:
        print("Apto a fazer recuperação")
else:
    print("Reprovado")

print("Fim do Programa")
```

File "<ipython-input-9-273742eb5b5b>", line 8

SyntaxError: '(' nunca foi fechado

SyntaxError: '(' was never closed

SEARCH STACK OVERFLOW



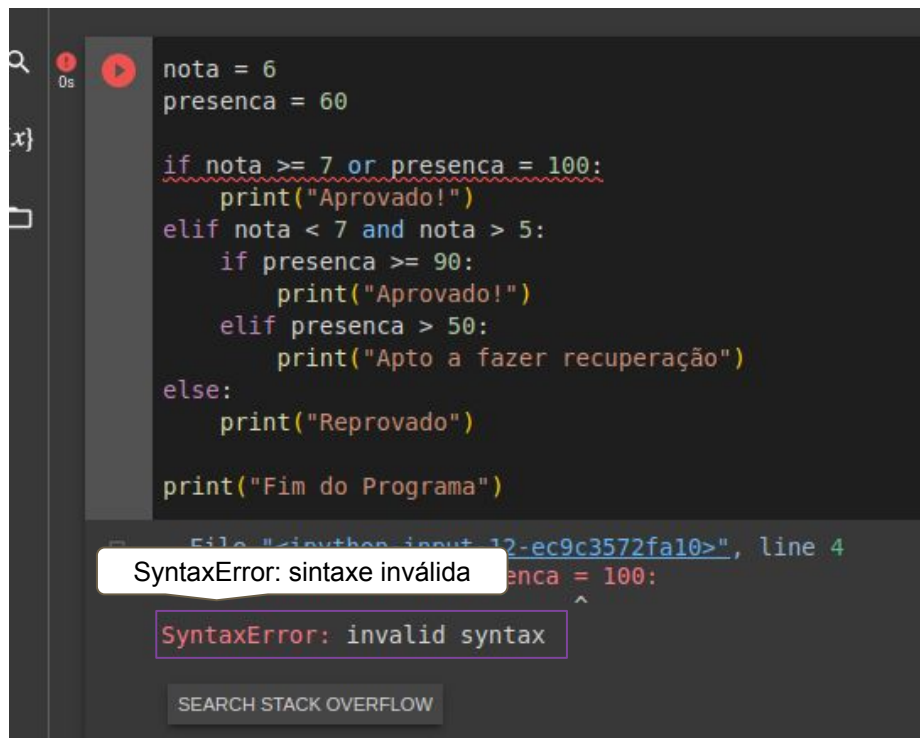
# Erros de sintaxe - Jogo dos 1 erros

```
▶ nota = 6
  presença = 60

  if nota >= 7 or presença = 100:
    print("Aprovado!")
  elif nota < 7 and nota > 5:
    if presença >= 90:
      print("Aprovado!")
    elif presença > 50:
      print("Apto a fazer recuperação")
  else:
    print("Reprovado")

  print("Fim do Programa")
```

# Erros de sintaxe - Jogo dos 1 erros



```
nota = 6
presenca = 60

if nota >= 7 or presenca = 100:
    print("Aprovado!")
elif nota < 7 and nota > 5:
    if presenca >= 90:
        print("Aprovado!")
    elif presenca > 50:
        print("Apto a fazer recuperação")
else:
    print("Reprovado")

print("Fim do Programa")
```

File "<input type="text" value="12-ec9c3572fa10">", line 4

**SyntaxError: invalid syntax**

SEARCH STACK OVERFLOW

# Erros de sintaxe - Jogo dos 1 erros

```
▶ nota = 6
  presenca = 60

  if nota >= 7 or presenca == 100:
    print("Aprovado!")
  elif nota < 7 and nota > 5:
    if presenca >= 90:
      print("Aprovado!")
    elif presenca > 50:
      print("Apto a fazer recuperação")
  else:
    print("Reprovado")

  print("Fim do Programa")
```

# Erros de sintaxe - Jogo dos 1 erros

```
nota = 6
presenca = 60

if nota >= 7 or presenca == 100:
    print("Aprovado!")
elif nota < 7 and nota > 5:
    if presenca >= 90:
        print("Aprovado!")
    eif presenca > 50:
        print("Apto a fazer recuperação")
else:
    print("Reprovado")

print("Fim do Programa")
```

File "<ipython-input-13-aa44e349fb29>", line 9

```
eif presenca > 50:
```

^

SyntaxError: invalid syntax

SEARCH STACK OVERFLOW

# Erros de sintaxe - Jogo dos 1 erros

```
▶ nota = 6
  presenca = 60

  if nota >= 7 or presenca == 100:
    print("Aprovado!")
  elif nota < 7 and nota > 5:
    if presenca >= 90:
      print("Aprovado!")
    elif presenca > 50:
      print("Apto a fazer recuperação")
  else:
    print("Reprovado")

  print("Fim do Programa")
```

# Erros de sintaxe - Jogo dos 1 erros

```
nota = 6
presenca = 60

if nota >= 7 or presenca == 100:
    print("Aprovado!")
elif nota < 7 and nota > 5:
    if presenca >= 90:
        print("Aprovado!")
    elif presenca > 50:
        print("Apto a fazer recuperação")
else:
    print("Reprovado")

print("Fim do Programa")
```

-----  
NameError Traceback (most recent call last)  
 <ipython-input-14-b9b230aecdbe> in <cell line: 4>()  
 7 if presenca >= 90:  
 8 print("Aprovado!")  
----> 9 elif presenca > 50:  
 10 print("Apto a fazer recuperação")  
 11 else:

NameError: name 'presenca' is not defined

SEARCH STACK OVERFLOW

# Erros de sintaxe - Jogo dos 1 erros

```
▶ nota = 6
  presença = 60

  if nota >= 7 or presença == 100:
    print("Aprovado!")
  elif nota < 7 and nota > 5:
    if presença >= 90:
      print("Aprovado!")
    elif presença > 50:
      print("Apto a fazer recuperação")
  else:
    print("Reprovado")

  print(["Fim do Programa"])
```

0s



```
nota = 6
presenca = 60

if nota >= 7 or presenca == 100:
    print("Aprovado!")
elif nota < 7 and nota > 5:
    if presenca >= 90:
        print("Aprovado!")
    elif presenca > 50:
        print("Apto a fazer recuperação")
else:
    print("Reprovado")

print("Fim do Programa")
```



File "<ipython-input-10-c42d62c3b388>", line 8

SyntaxError: é esperado um bloco indentado após a cláusula if na linha 7

IndentationError: expected an indented block after 'if' statement on line 7

SEARCH STACK OVERFLOW



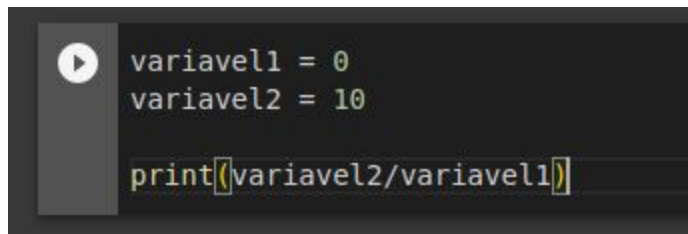
# Erros de sintaxe - indentação

- Erros de indentação também são erros de sintaxe em Python.
- Como a indentação é obrigatória, o código só poderá ser compreendido com a indentação correta.
- Porém, esse tipo de erro levanta uma exceção própria. Ela é muito comum em python mas não ocorre em outras linguagens que usam outros marcadores de sintaxe.

# Erros de execução

- Outro tipo de ocorre quando o código em si está escrito de forma correta, porém algo ocorre durante a execução que impede que o código seja interpretado.

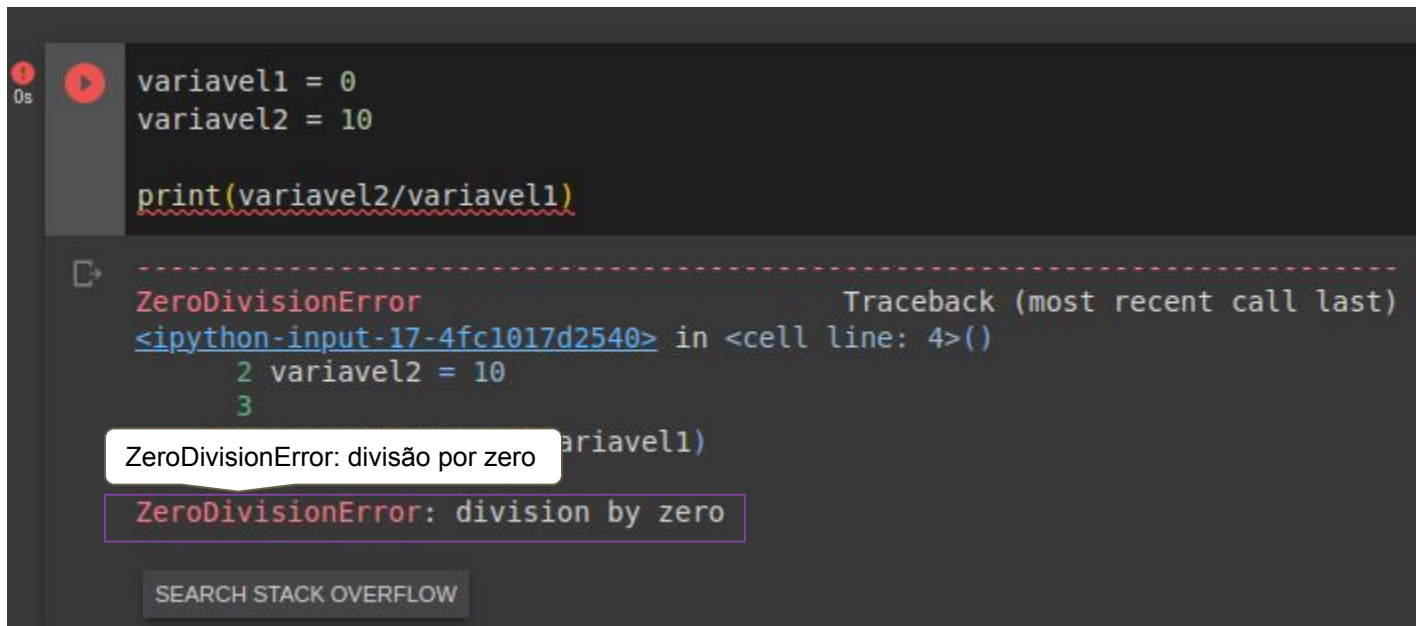
# Erros de execução



```
variavel1 = 0  
variavel2 = 10  
  
print(variavel2/variavel1)
```

The image shows a code editor with a dark background. On the left, there is a vertical grey bar with a white play button icon. To the right of this bar, the code is written in a light green monospace font. The code consists of three lines: 'variavel1 = 0', 'variavel2 = 10', and 'print(variavel2/variavel1)'. The third line is highlighted with a yellow background.

# Erros de execução



The screenshot shows a Jupyter Notebook interface. At the top, there is a red play button icon and a timer showing '0s'. Below this, the code is written in a dark-themed editor:

```
variavel1 = 0  
variavel2 = 10  
  
print(variavel2/variavel1)
```

The line `print(variavel2/variavel1)` is underlined with a red wavy line, indicating an error. Below the code editor, there is a traceback window showing the error details:

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-17-4fc1017d2540> in <cell line: 4>()  
      2 variavel2 = 10  
      3  
      4 print(variavel2/variavel1)  
ZeroDivisionError: division by zero
```

A tooltip is visible over the error message, displaying the text: "ZeroDivisionError: divisão por zero". At the bottom of the notebook interface, there is a button labeled "SEARCH STACK OVERFLOW".



```
variavel1 = 0  
variavel2 = 10  
  
print(variavel2/variavel3)
```

0s



```
variavel1 = 0  
variavel2 = 10
```


```
print(variavel2/variavel3)
```



```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-19-d7df919ccd15> in <cell line: 4>()  
      2 variabel2 = 10  
      3  
----> 4 print(variavel2/variavel3)
```

```
NameError: name 'variavel3' is not defined
```

SEARCH STACK OVERFLOW

- 
- Existem muitas outras exceções e elas podem ser entendidas na documentação da linguagem:  
<https://docs.python.org/3/library/exceptions.html>

5 Minutinhos pra resfriar o cérebro  
Tomem água!





# Erros semânticos

- Erros semânticos (pra mim) são os mais difíceis de todos.
- O código está sintaticamente correto e não ocorre erro na execução.
- Porém o resultado obtido não é o resultado esperado.
- Isso quer dizer que existe um erro na lógica do programa.
- Vamos retomar o exemplo das notas para discutir esse tipo de erro.

## Erros semânticos - Retomando o exemplo

- Vamos supor que o professor agora criou uma regra super complicada para aprovação por presença.
- Se o aluno atingiu a média ou 100% da presença, está aprovado.
- Se o aluno não atingiu a média para aprovação, mas tiver nota acima de 5 e 90% de presença, está aprovado.
- Se o aluno tiver nota acima de 6, e tiver presença acima de 50%, ele tem direito a recuperação.
- Se não tiver nem nota acima de 6, nem presença acima de 50%, o aluno está reprovado.

# Erros semânticos

```
aula3 > erros > errosemanico.py > ...
1  nota = 6
2  presenca = 50
3
4  if nota >= 7 or presenca == 100:
5      print("Aprovado!")
6  elif nota < 7 and nota > 5:
7      if presenca >= 90:
8          print("Aprovado!")
9      elif presenca > 50:
10         print("Apto a fazer recuperação")
11 else:
12     print("Reprovado")
13
14 print("Fim do Programa")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS bash - erros

• belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/erros\$ python3 errosemanico.py  
Fim do Programa

# Debugging

- Para identificar mais facilmente onde o erro está ocorrendo, nós podemos usar uma ferramenta chamada debugger.
- Ela permite que a execução seja observada linha por linha, observando os valores das nossas variáveis durante a execução.
- O debugger não é a única forma de debugging possível, mas é bastante utilizada e muito útil.
- Vamos aprender a usar o debugger do vscode:

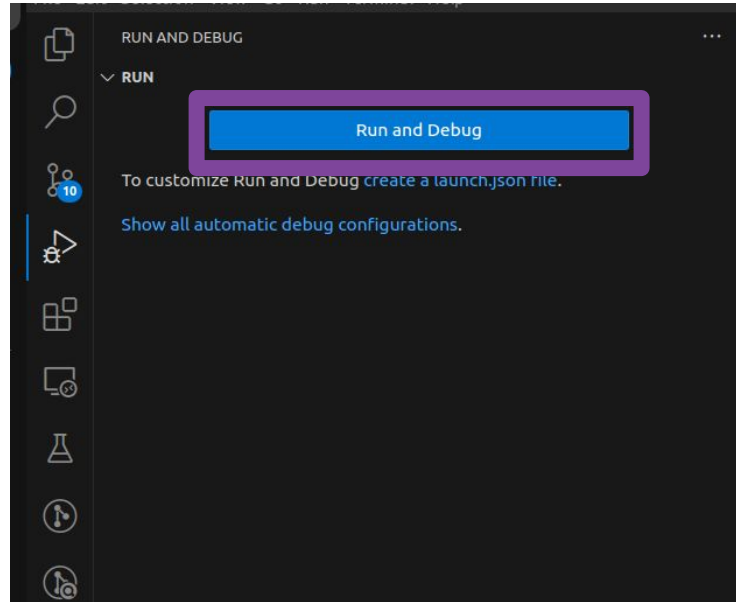
# Usando o debugger - Adicionando Breakpoint

```
aula3 > erros > errosementico.py > ...  
1  nota = 6  
2  presenca = 50  
3  
• 4  if nota >= 7 or presenca == 100:  
5      print("Aprovado!")  
6  elif nota < 7 and nota > 5:  
7      if presenca >= 90:  
8          print("Aprovado!")  
9      elif presenca > 50:  
10         print("Apto a fazer recuperação")  
11  else:  
12      print("Reprovado")  
13  
14  print("Fim do Programa")
```

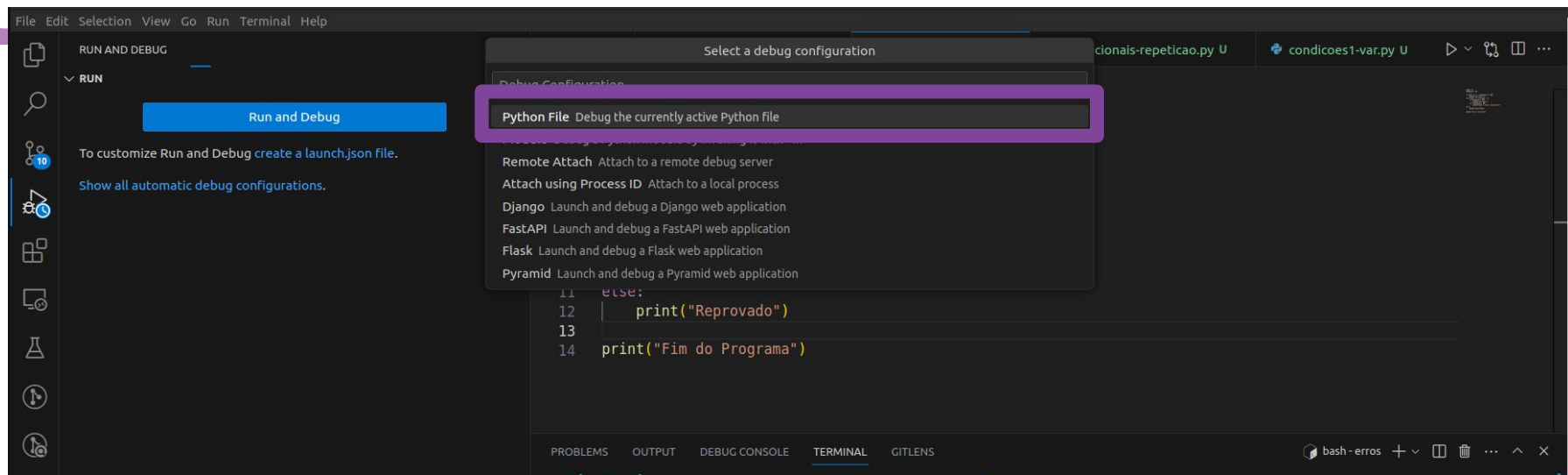
# Usando o debugger



# Usando o debugger



# Usando o debugger



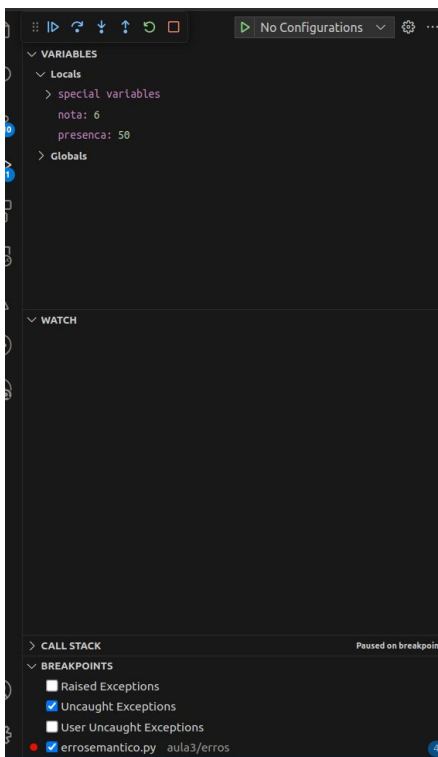


# Usando o debugger

aula3 > erros > errosemantico.py > ...

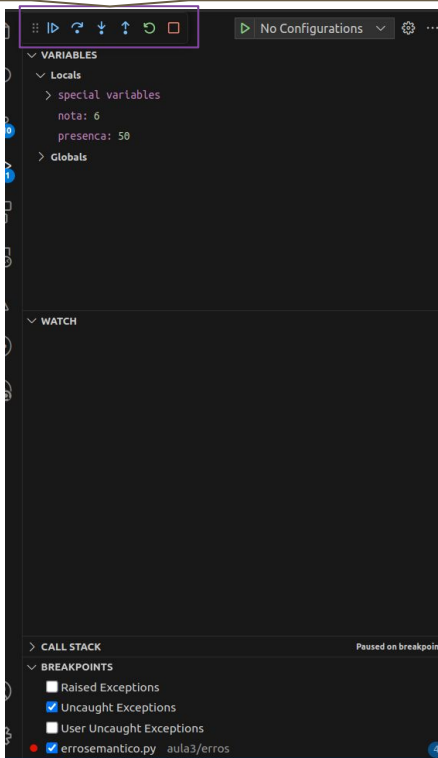
```
1  nota = 6
2  presenca = 50
3
4  if nota >= 7 or presenca == 100:
5      print("Aprovado!")
6  elif nota < 7 and nota > 5:
7      if presenca >= 90:
8          print("Aprovado!")
9      elif presenca > 50:
10         print("Apto a fazer recuperação")
11 else:
12     print("Reprovado")
13
14 print("Fim do Programa")
```

# Usando o debugger

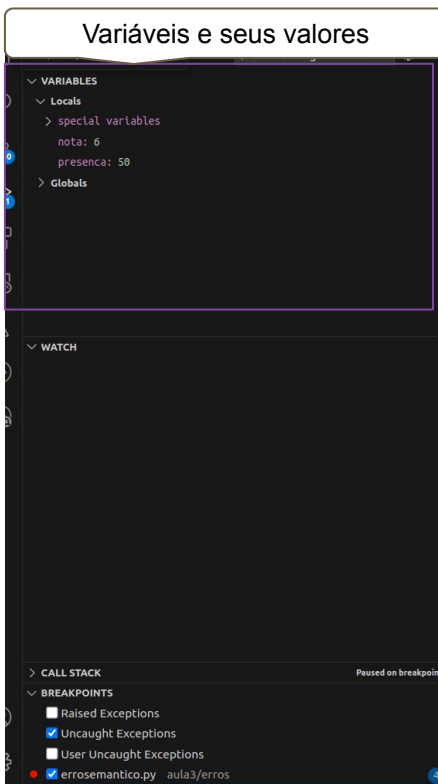


# Usando o debugger

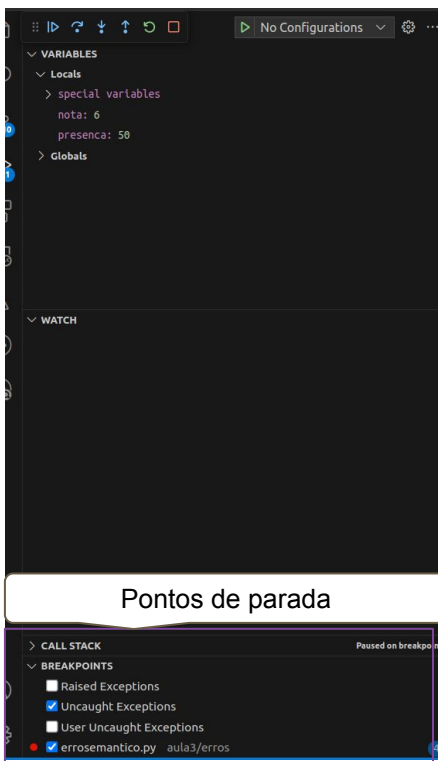
Ferramenta para pausar e avançar a execução



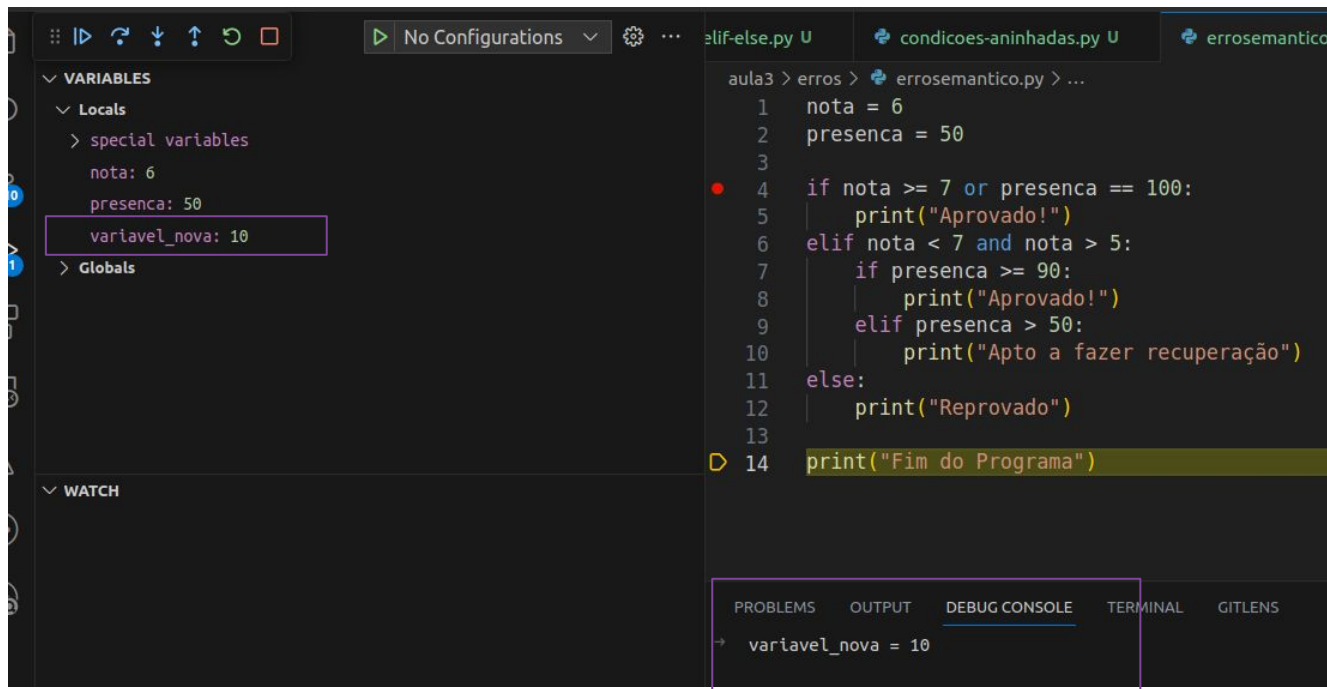
# Usando o debugger



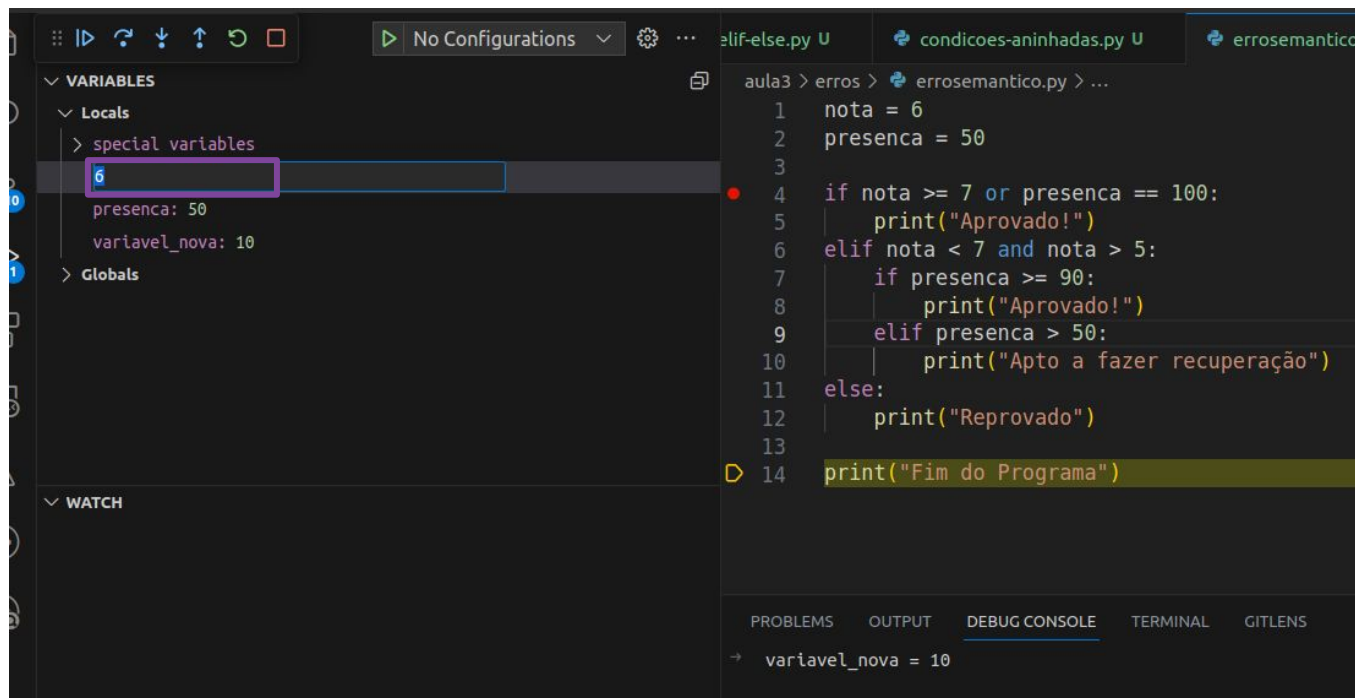
# Usando o debugger



# Usando o debugger - Console



# Usando o debugger - Editando valor durante execução



5 minutinhos pra refletir - Tomem água!





# Tratando exceções

- Nós podemos ter controle sobre erros que sabemos que podem acontecer.
- Por exemplo, se queremos fazer uma divisão por dois números informados pelo usuário e sabemos que divisão por zero causa um erro que encerra o programa prematuramente, nós podemos impedir que a divisão ocorra.
- Para isso, usaremos as expressões **Try** e **Except**
- Elas podem ser traduzidas como Tente e Excessão, respectivamente.
- Vamos ao exemplo:

# Tratando exceções - Código

```
aula3 > erros > try-catch.py > numero2
1  numero1 = int(input())
2  numero2 = int(input())
3
4  print(numero1/numero2)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

```
belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/erros$ python3 try-catch.py
1
3
0.3333333333333333
belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/erros$
```

# Tratando exceções - Exceção ocorre

```
aula3 > erros > try-catch.py > numero2
```

```
1  numero1 = int(input())
2  numero2 = int(input())
3
4  print(numero1/numero2)
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

GITLENS

```
belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/erros$ python3 try-catch.py
```

```
1
```

```
0
```

```
Traceback (most recent call last):
```

```
  File "/home/belisa/Documents/reprograma/aula3/erros/try-catch.py", line 4, in <module>
    print(numero1/numero2)
```

```
ZeroDivisionError: division by zero
```

# Tratando exceções - Tratando o erro previamente

```
aula3 > erros > try-catch.py > ...
```

```
3
4 try:
5     print(numero1/numero2)
6 except:
7     print("Não é possível dividir por zero")
8
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL GITLENS

```
belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/erros$ python3 try-catch.py
1
0
Não é possível dividir por zero
belisa@belisa-Vostro-3520:~/Documents/reprograma/aula3/erros$ _
```

