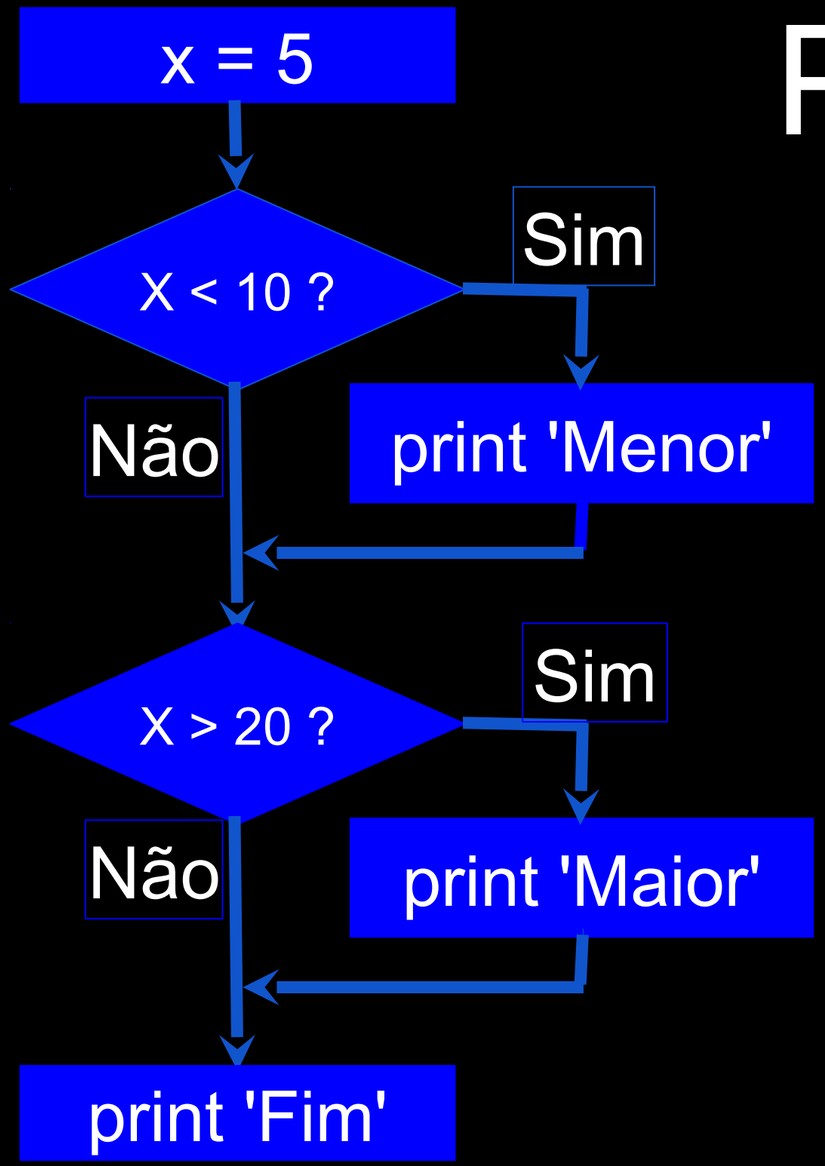
Execução Condicional

Prof. Alberto Costa Neto

Programação em Python

Passos Condicionais



Programa:

```
x = 5
if x < 10:
    print 'Menor'

if x > 20:
    print 'Maior'

print 'Fim'
```

Saída:

```
Menor
Fim
```

Operadores de Comparação

- Expressões booleanas fazem uma pergunta e produzem Sim (Verdadeiro) ou Não (Falso). Usamos este resultado para controlar o fluxo do programa.
- Expressões booleanas usando operadores de comparação resultam em – True/False – Sim/Não
- Operadores de comparação consultam as variáveis e valores mas não os modificam

Python	Significado
<	Menor que
<=	Menor ou igual a
==	Igual a
>=	Maior ou igual a
>	Maior que
!=	Diferente

Lembrete: "=" é usado para atribuição.
Não confunda!

http://en.wikipedia.org/wiki/George_Boole

Operadores de Comparação

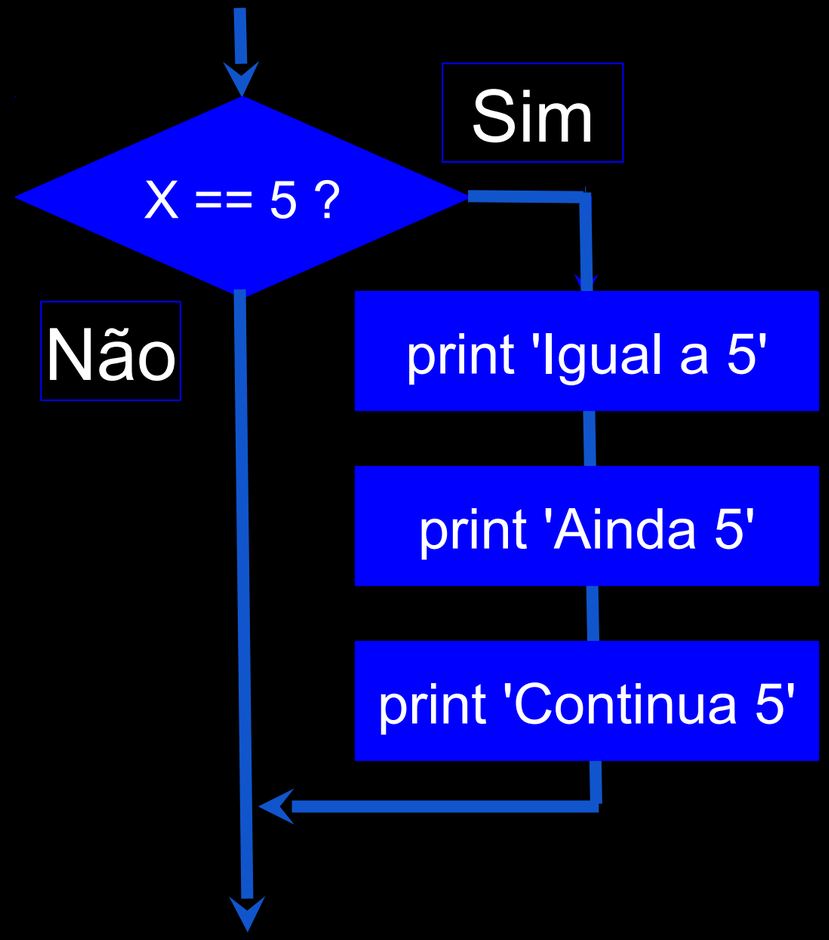
```
x = 5
if x == 5 :
    print 'Igual a 5'
if x > 4 :
   print 'Maior que 4'
if  x >= 5 :
    print 'Maior ou igual a 5'
if x < 6 : print 'Menor que 6'
if x <= 5 :
    print 'Menor que ou igual a 5'
if x != 6 :
    print 'Diferente de 6'
```

```
Igual a 5
Maior que 4
Maior ou igual a 5
Menor que 6
Menor ou igual a 5
Diferente de 6
```

Decisões simples

```
x = 5
print 'Antes de 5'
if  x == 5 :
    print 'Igual a 5'
    print 'Ainda 5'
    print 'Continua 5'
print 'Depois de 5'
print 'Antes de 6'
if x == 6 :
    print 'Igual a 6'
    print 'Ainda 6'
    print 'Continua 6'
print 'Depois de 6'
```

```
Antes de 5
Igual a 5
Ainda 5
Continua 5
Depois de 5
Antes de 6
Depois de 6
```



Indentação

- Aumentar a indentação depois de comandos if ou for (a seguir)
- Mantenha a indentação para indicar o escopo do bloco (quais linhas são afetadas pelo if/for)
- Reduza a indentação *de volta ao* nível do comando if ou for para indicar o fim do bloco de comandos
- Linhas em branco são ignoradas – não afetam em nada a indentação
- Comentários em uma linha por sim só são ignorados no que diz respeito a indentação

Alerta: Não use Tabulação

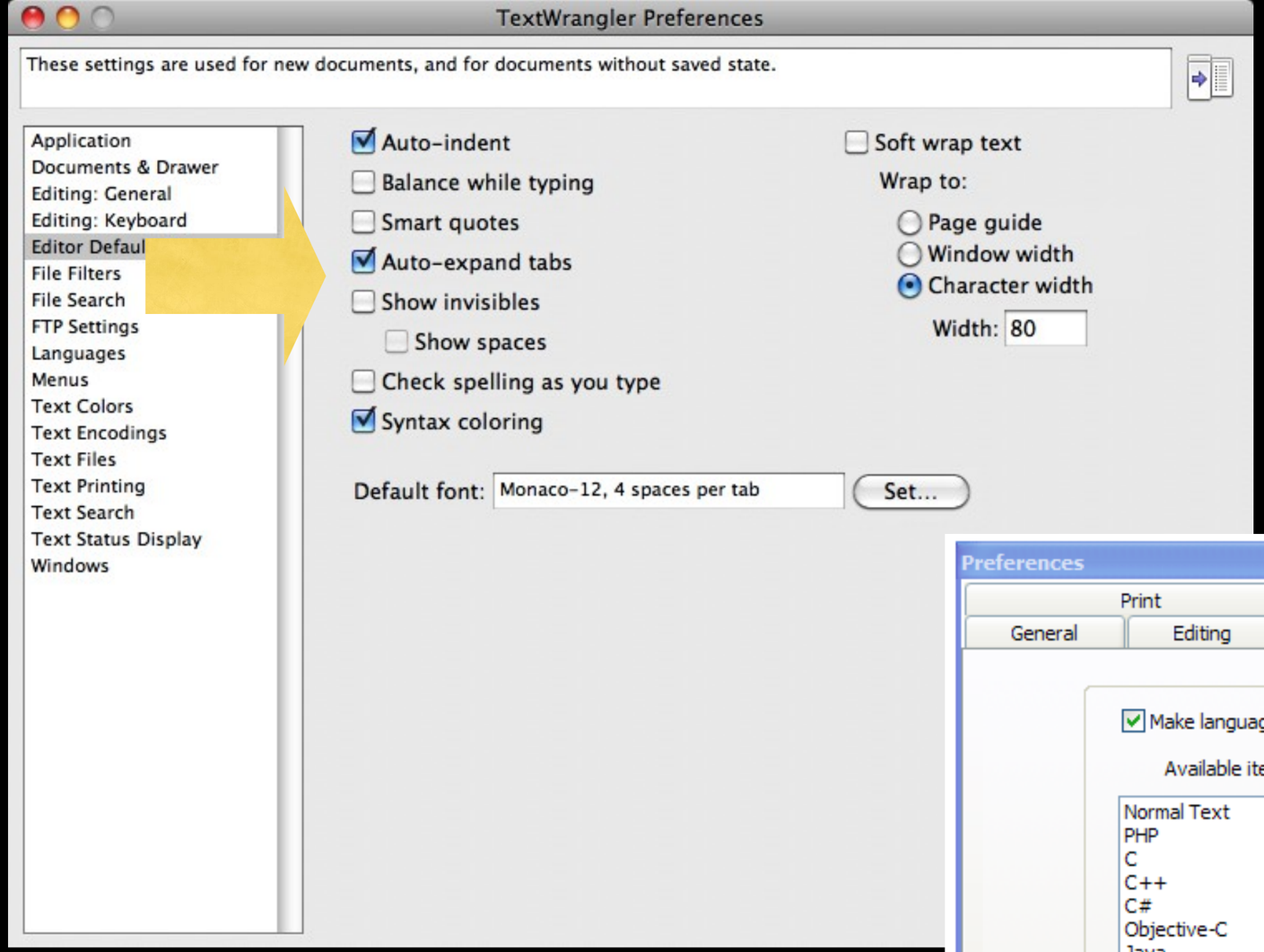
- A maioria dos editores de texto pode transformar tabulação em espaços - tenha certeza que habilitou esta funcionalidade no editor

  > NotePad++: Configurações -> Preferências -> Configurações do TAB

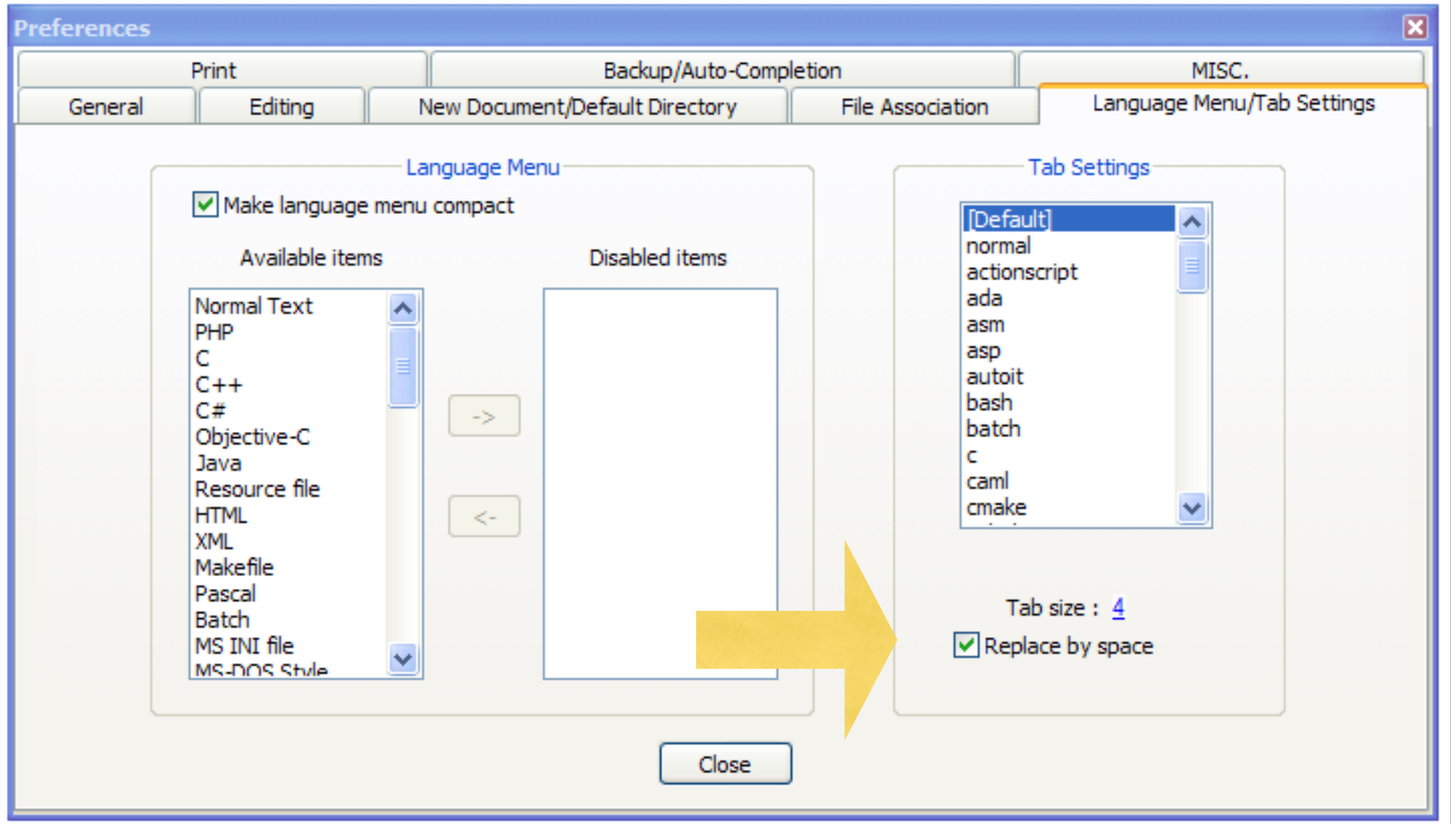
  > TextWrangler: TextWrangler -> Preferences -> Editor Defaults

  > SublimeText: View → Indentation → Indent using spaces

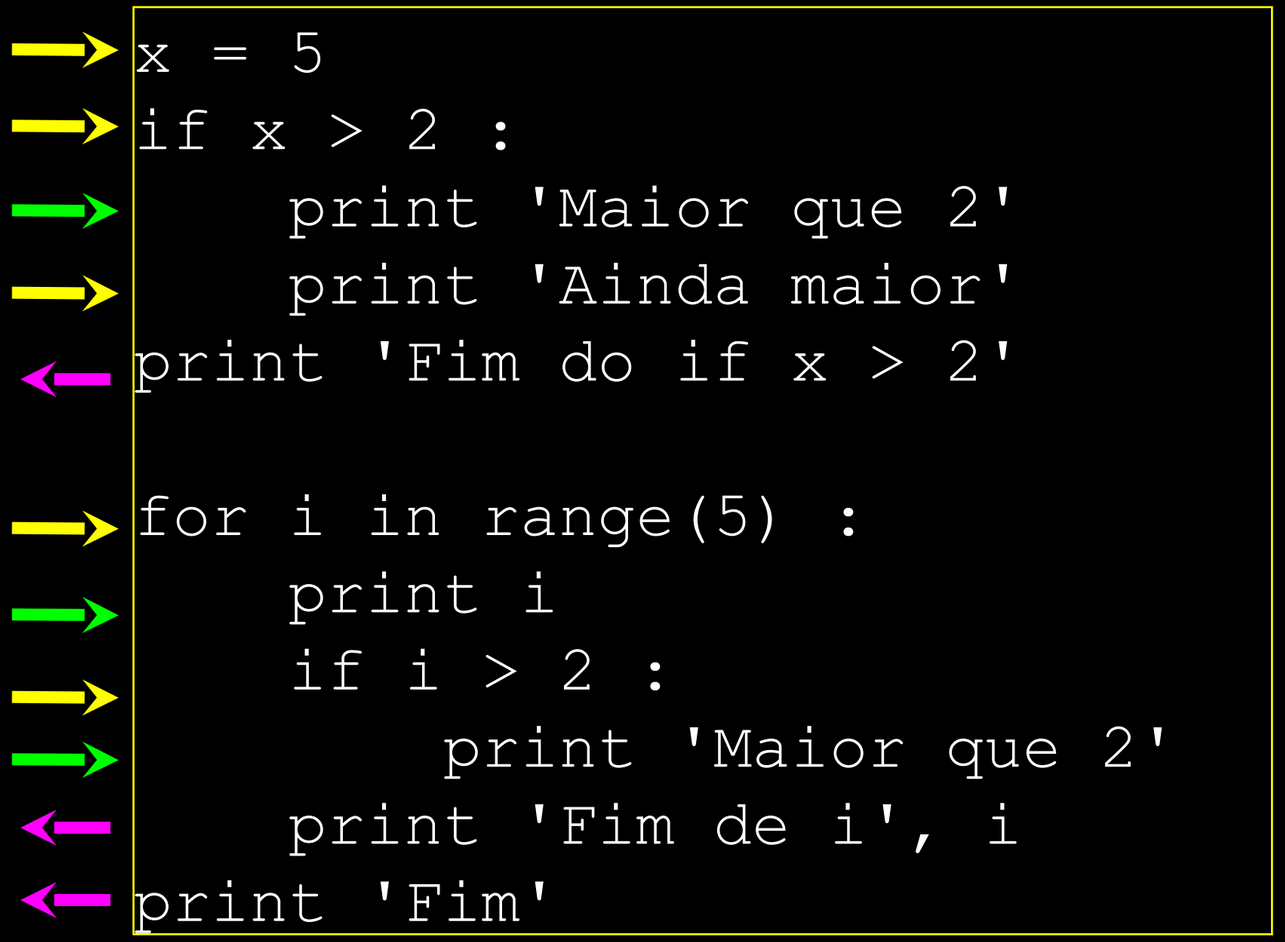
- Python dá *muita* importância à indentação. Um caractere a mais ou a menos muda o escopo. Se você misturar tabulação e espaços, provavelmente terá “indentation errors” mesmo que aparentemente o código esteja correto



Essa simples configuração irá lhe poupar horas de raiva!



Aumentar depois de if ou for para
incluir os comandos dentro deles
Diminua para indicar o fim do bloco



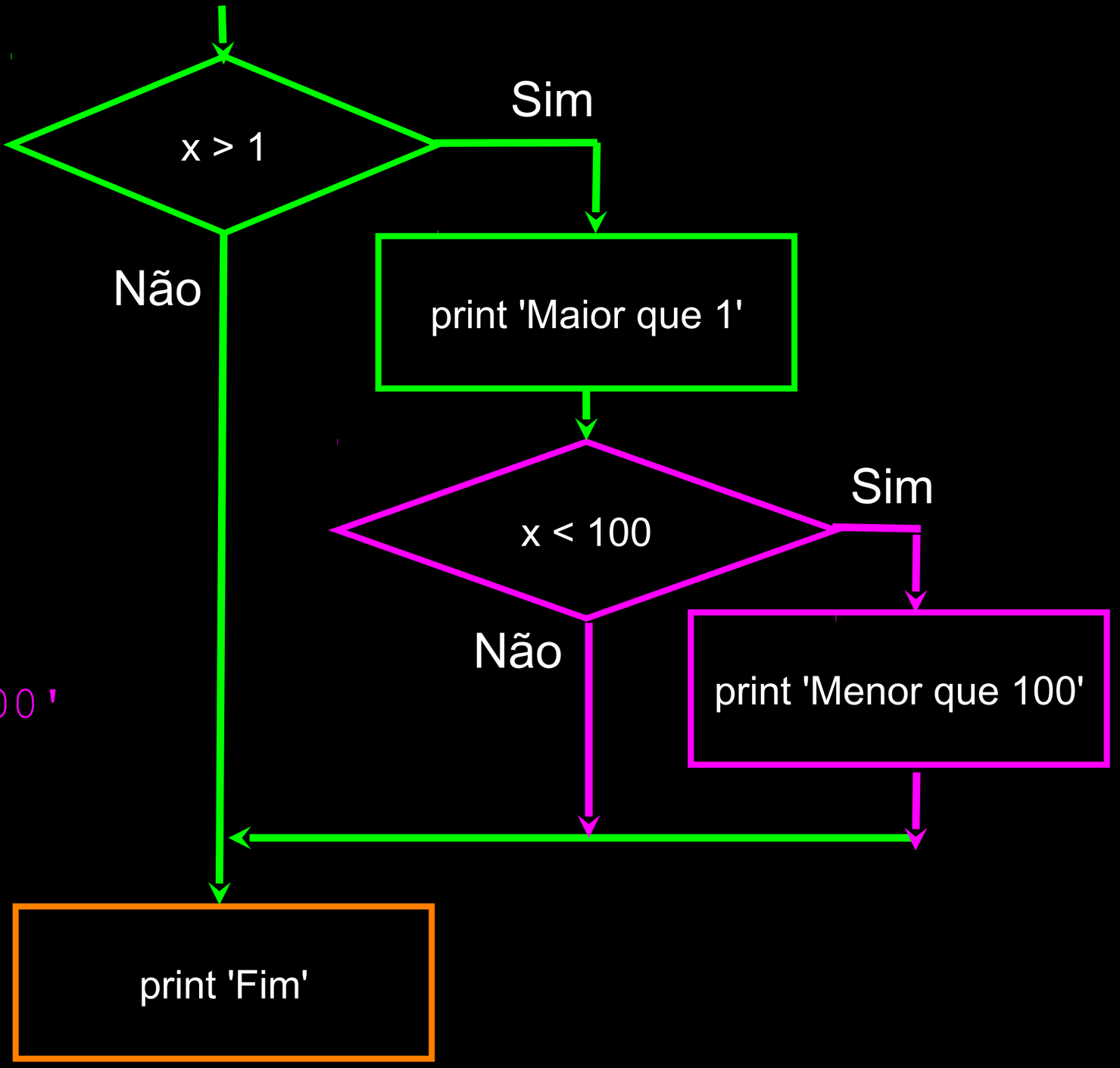
```
x = 5
if x > 2 :
    print 'Maior que 2'
    print 'Ainda maior'
print 'Fim do if x > 2'

for i in range(5) :
    print i
    if i > 2 :
        print 'Maior que 2'
    print 'Fim de i', i
print 'Fim'
```

Raciocinando sobre o início e fim dos blocos

```
x = 5
if x > 2 :
    print 'Maior que 2'
    print 'Ainda maior'
print 'Fim do if x > 2'

for i in range(5) :
    print i
    if i > 2 :
        print 'Maior que 2'
    print 'Fim de i', i
print 'Fim'
```

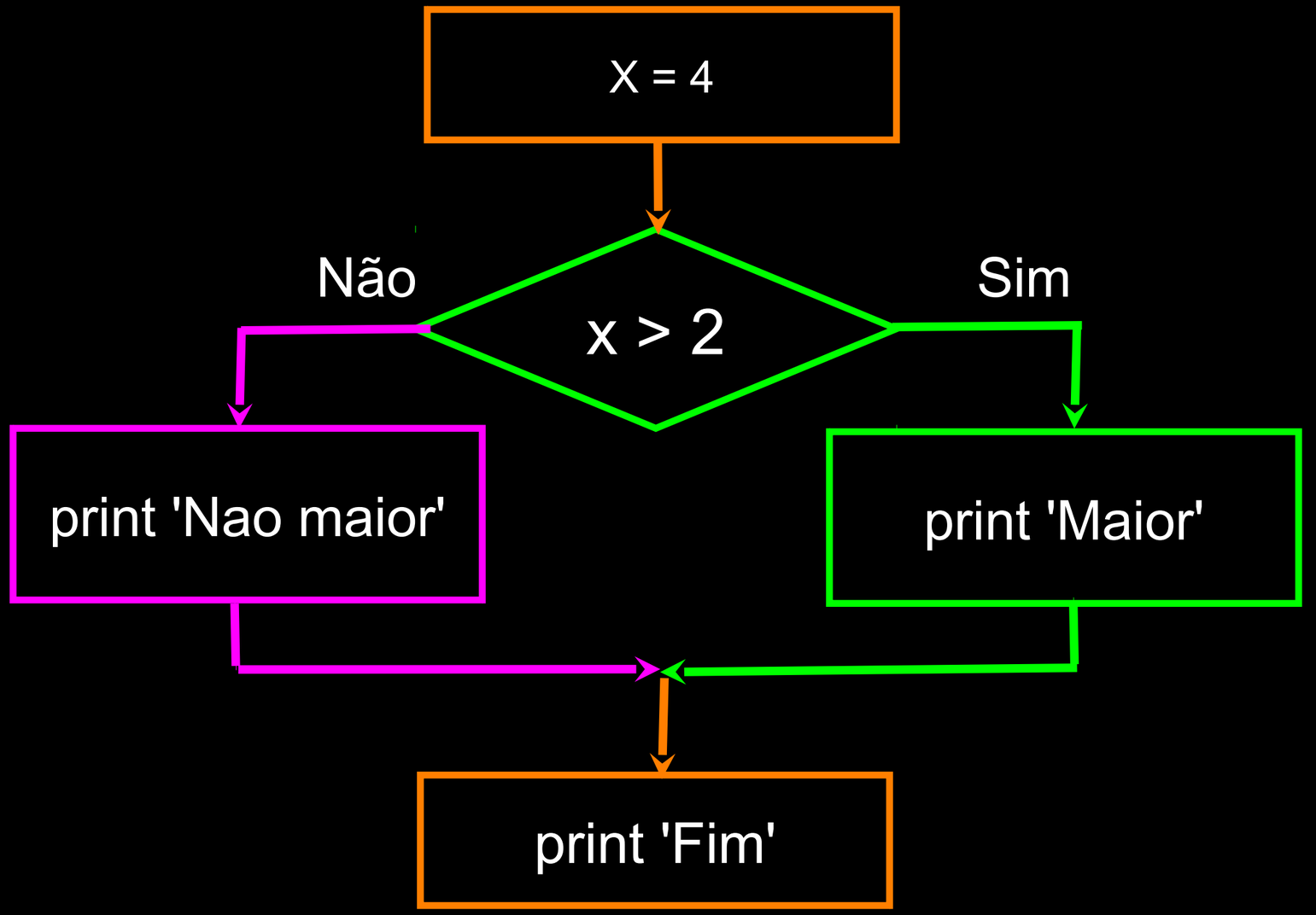
Decisões Aninhadas

```
x = 42
if x > 1 :
    print 'Maior que 1'
    if x < 100 :
        print 'Menor que 100'
print 'Fim'
```



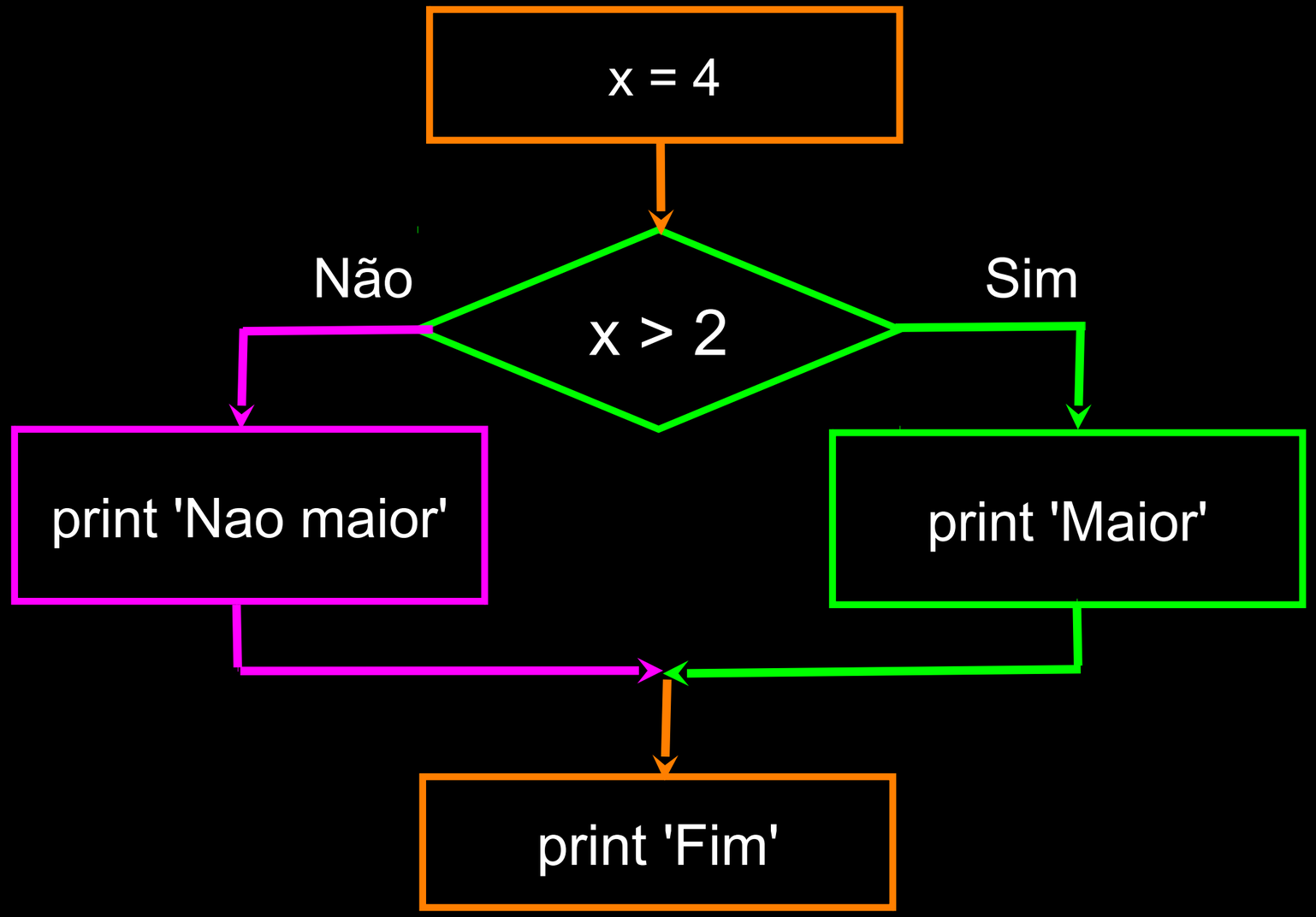
Decisões de caminho duplo

- Algumas vezes queremos fazer uma coisa se uma expressão lógica for verdadeira e outra coisa se for falsa
- É como uma bifurcação em uma estrada – temos que escolher um ou outro caminho, mas não ambos



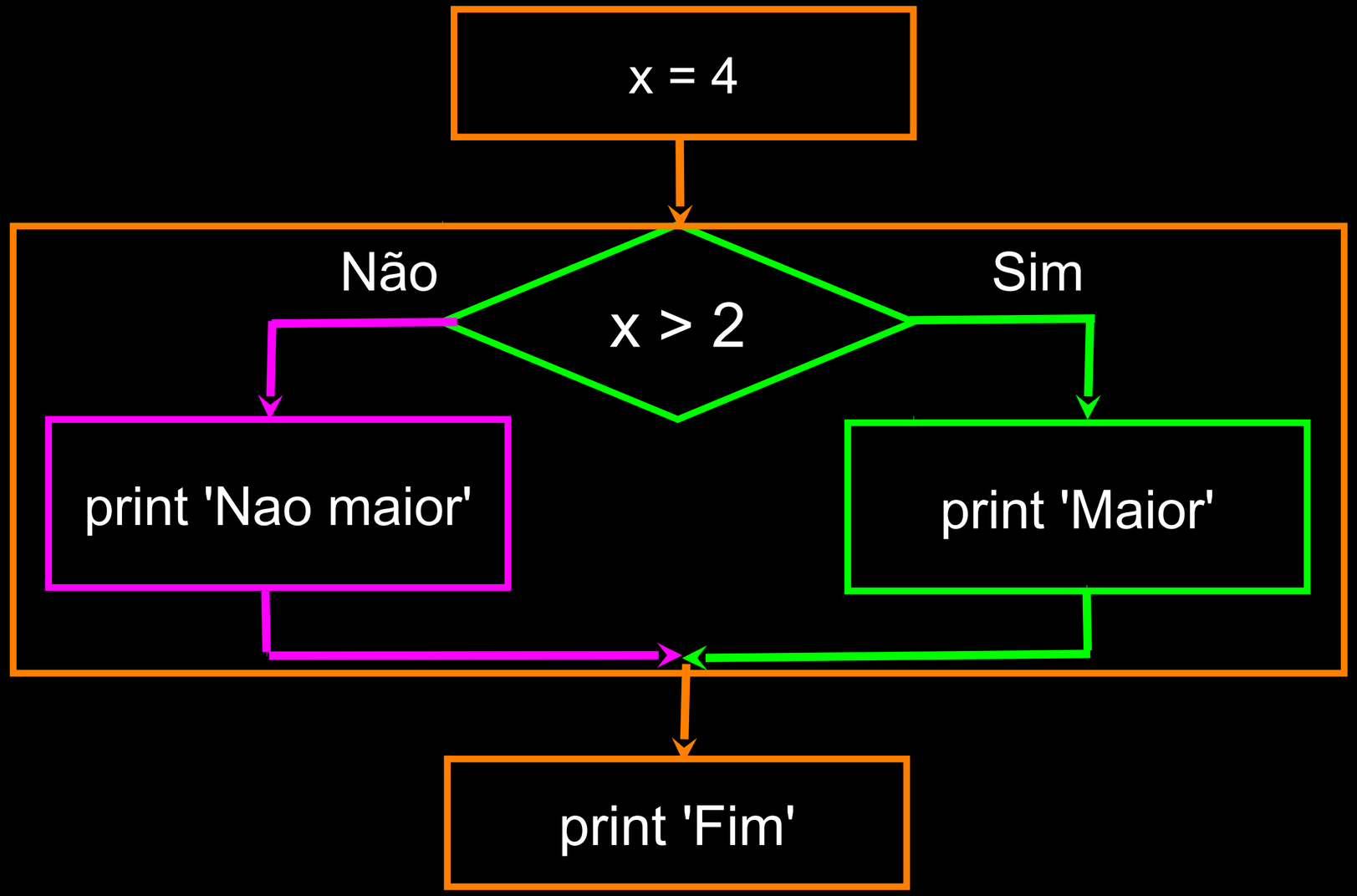
Caminho duplo com else :

```
x = 4

if x > 2 :
    print 'Maior'
else :
    print 'Nao maior'

print 'Fim'
```



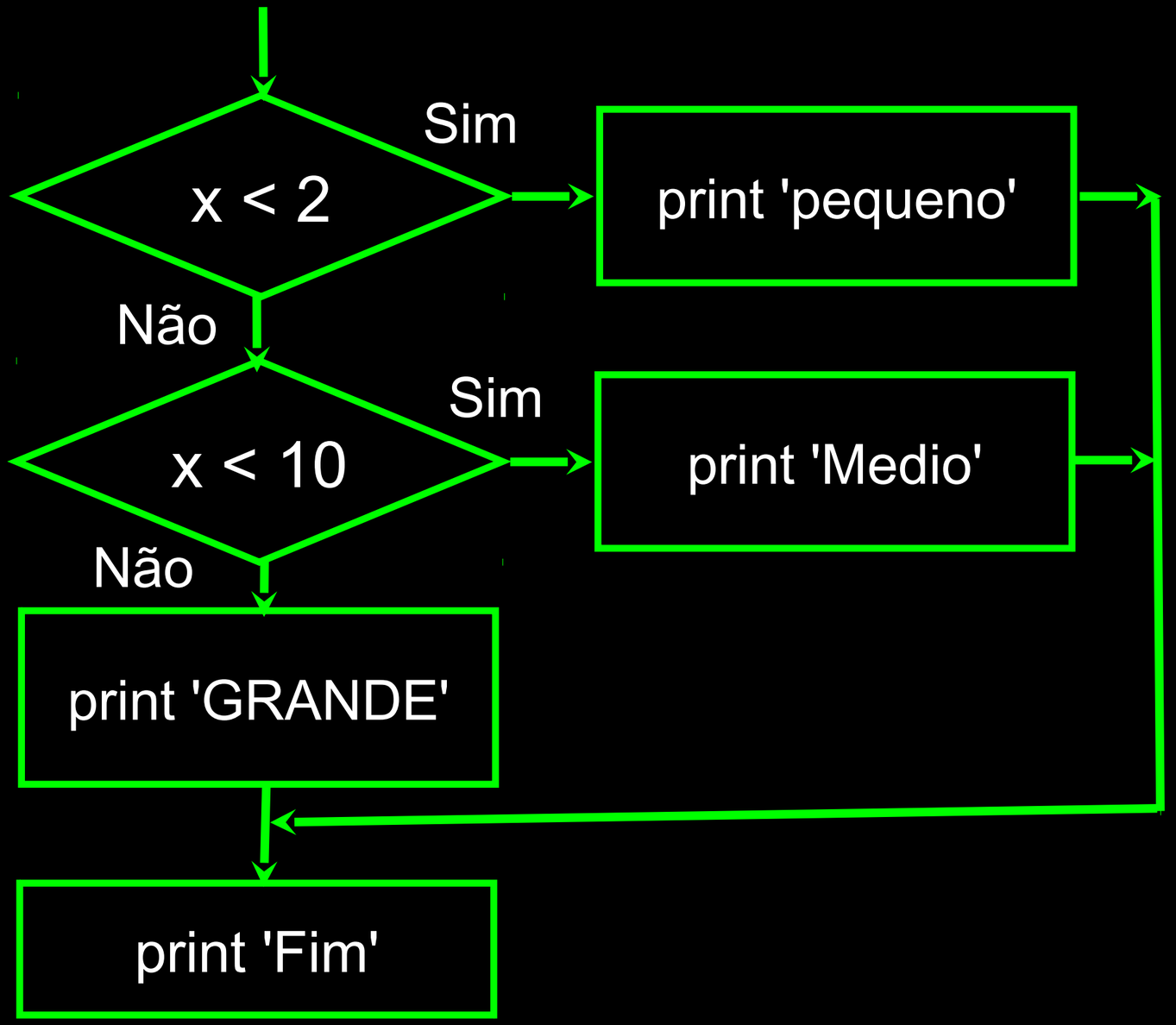
Caminho duplo com else :

```
x = 4

if x > 2 :
    print 'Maior'
else :
    print 'Nao maior'

print 'Fim'
```



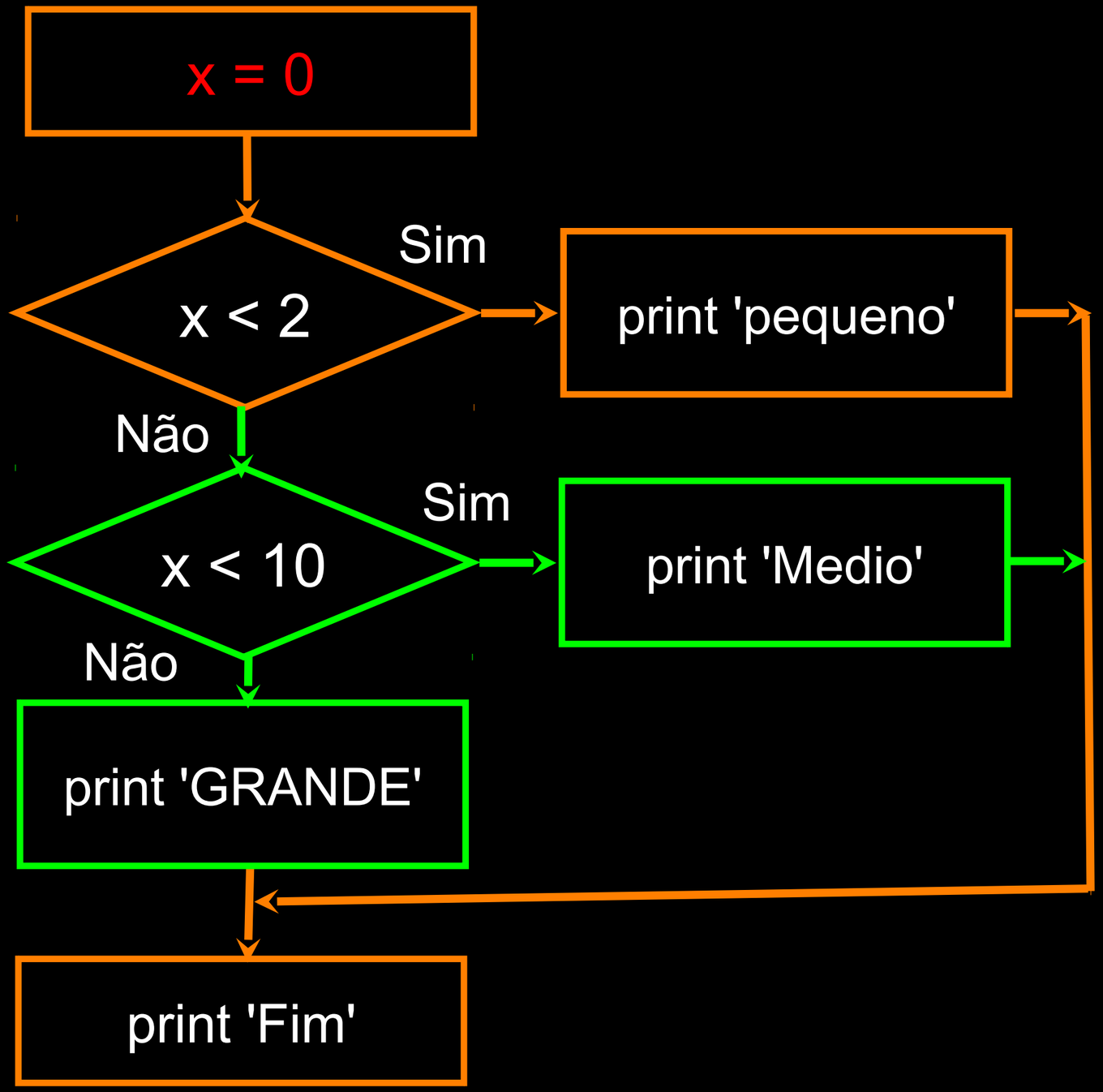
Caminhos Múltiplos

```
if x < 2 :
    print 'pequeno'
elif x < 10 :
    print 'Medio'
else :
    print 'GRANDE'
print 'Fim'
```



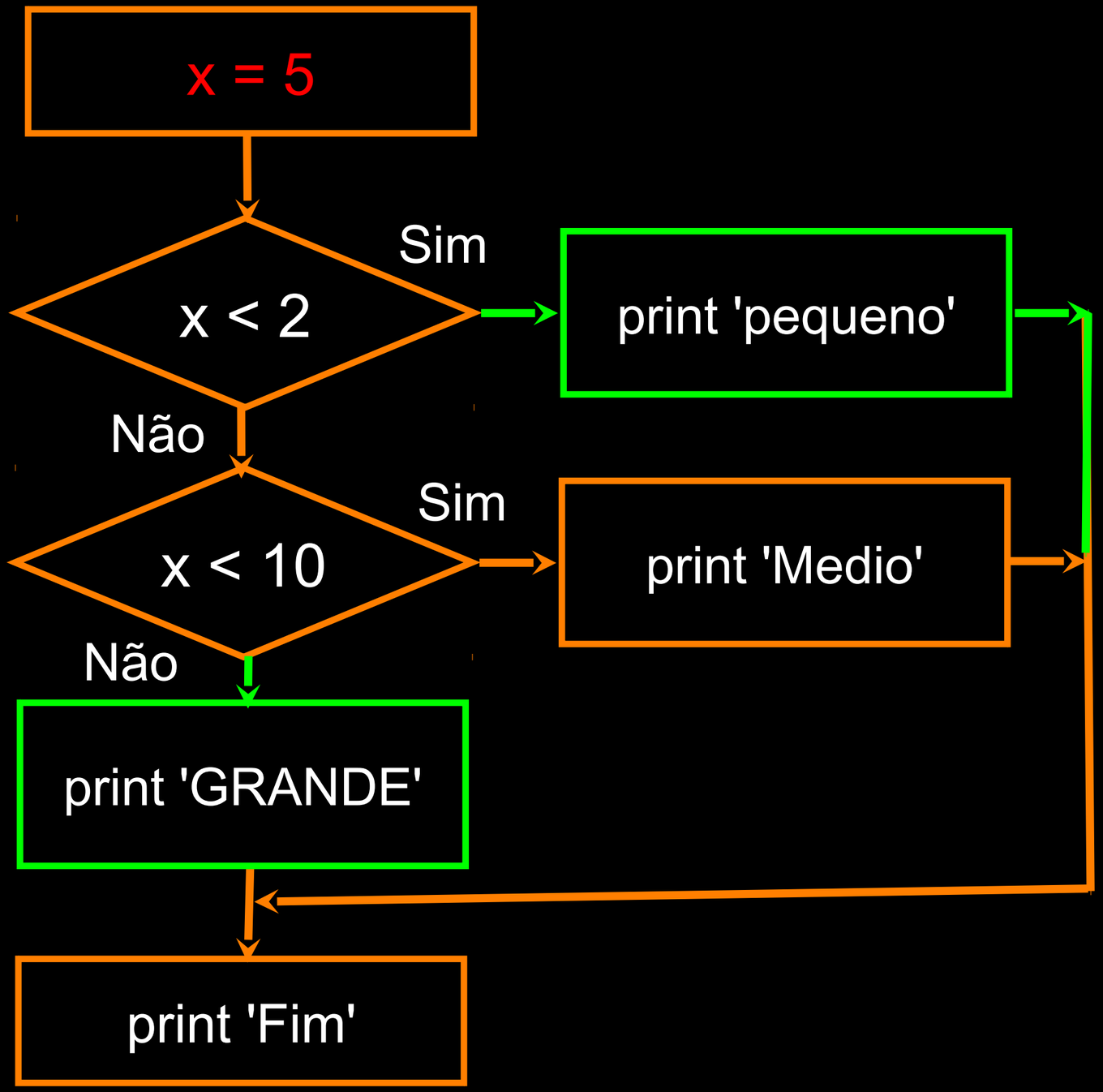
Caminhos Múltiplos

```
x = 0
if x < 2 :
    print 'pequeno'
elif x < 10 :
    print 'Medio'
else :
    print 'GRANDE'
print 'Fim'
```



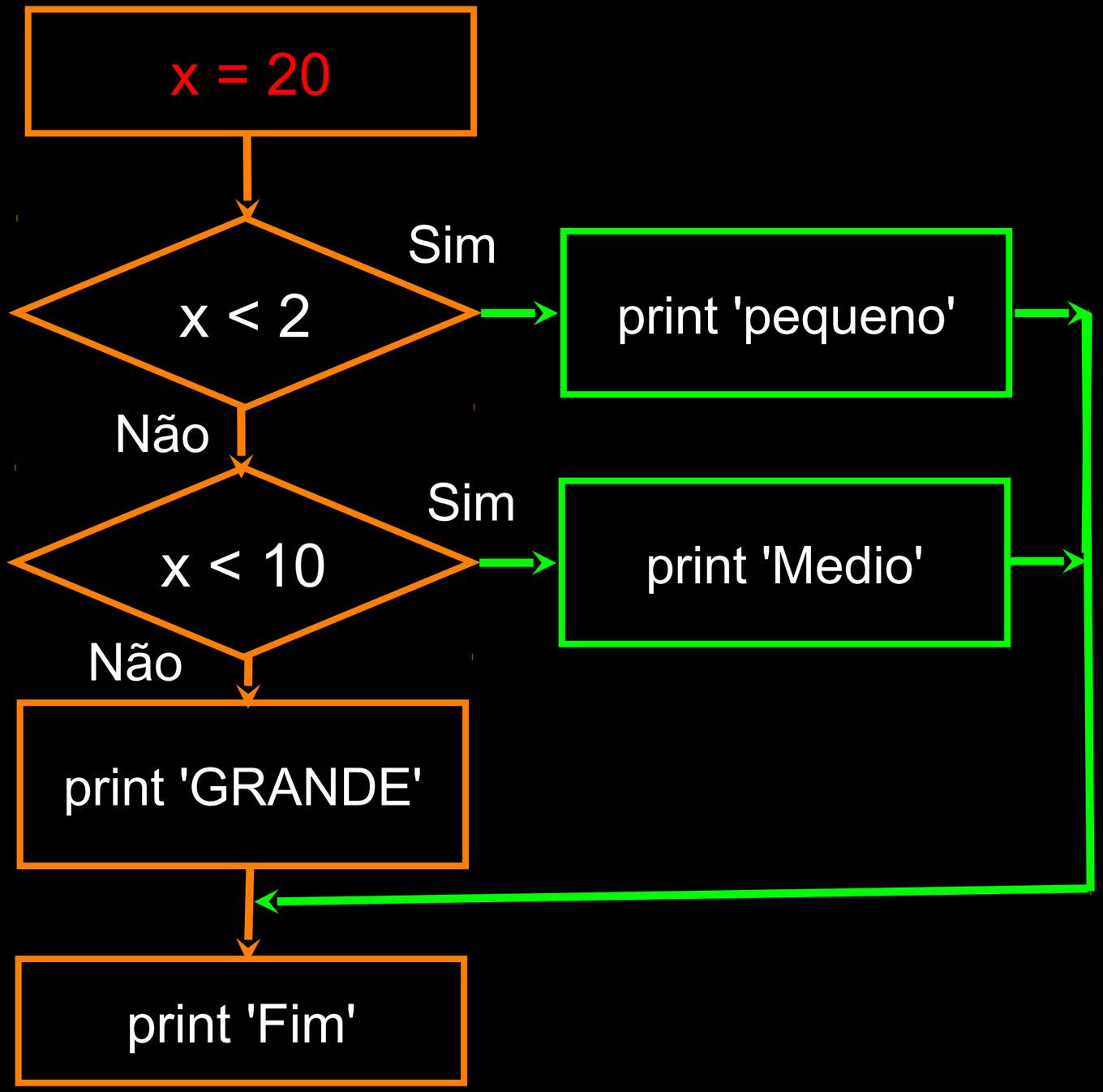
Caminhos Múltiplos

```
x = 5
if x < 2 :
    print 'pequeno'
elif x < 10 :
    print 'Medio'
else :
    print 'GRANDE'
print 'Fim'
```



Caminhos múltiplos

```
x = 20
if x < 2 :
    print 'pequeno'
elif x < 10 :
    print 'Medio'
else :
    print 'GRANDE'
print 'Fim'
```



Caminhos múltiplos

```
# Sem else
x = 5
if x < 2 :
    print 'pequeno'
elif x < 10 :
    print 'Medio'

print 'Fim'
```

```
if x < 2 :
    print 'pequeno'
elif x < 10 :
    print 'Medio'
elif x < 20 :
    print 'Grande'
elif x < 40 :
    print 'Enorme'
elif x < 100:
    print 'Gigante'
else :
    print 'Estupidamente Grande'
```

Quebra-cabeças múltiplos

Quais mensagens nunca serão impressas?

```
if x < 2 :
    print 'Menor que 2'
elif x >= 2 :
    print '2 ou mais'
else :
    print 'Outra coisa'
```

```
if x < 2 :
    print 'Menor que 2'
elif x < 20 :
    print 'Menor que 20'
elif x < 10 :
    print 'Menor que 10'
else :
    print 'Outra coisa'
```

A estrutura try / except

- Você pode colocar um bloco de comandos dentro um try/except
- Se o código dentro do try funcionar – o que está em except é desconsiderado
- Se o código dentro do try gerar um erro – pula-se para os comandos dentro da seção except

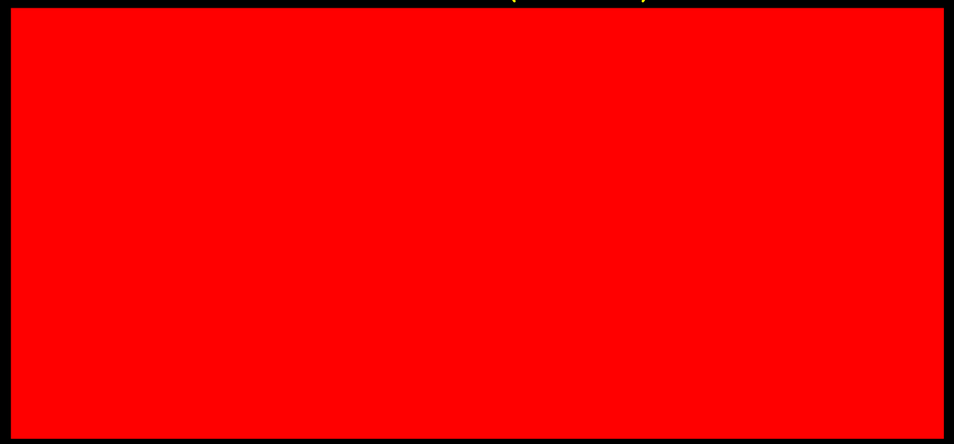
```
$ cat notry.py
str = 'Ola'
inteiro = int(str)
print 'Primeiro', inteiro
str = '123'
inteiro = int(str)
print 'Segundo', inteiro
```

```
$ python notry.py
Traceback (most recent call last):
File "notry.py", line 2, in <module>
 inteiro = int(str)
ValueError: invalid literal for int()
with base 10: 'Olá'
```

Fim

```
$ cat notry.py
str = 'Ola'
inteiro = int(str)
```

O programa para aqui



```
$ python notry.py
Traceback (most recent call last):
File "notry.py", line 2, in <module>
inteiro = int(str)
ValueError: invalid literal for int()
with base 10: 'Olá'
```

Fim

```
$ cat tryexcept.py
str = 'Ola Bob'
try:
    inteiro = int(str)
except:
    inteiro = -1

print 'Primeiro', str

str = '123'
try:
    inteiro = int(str)
except:
    inteiro = -1

print 'Segundo', inteiro
```

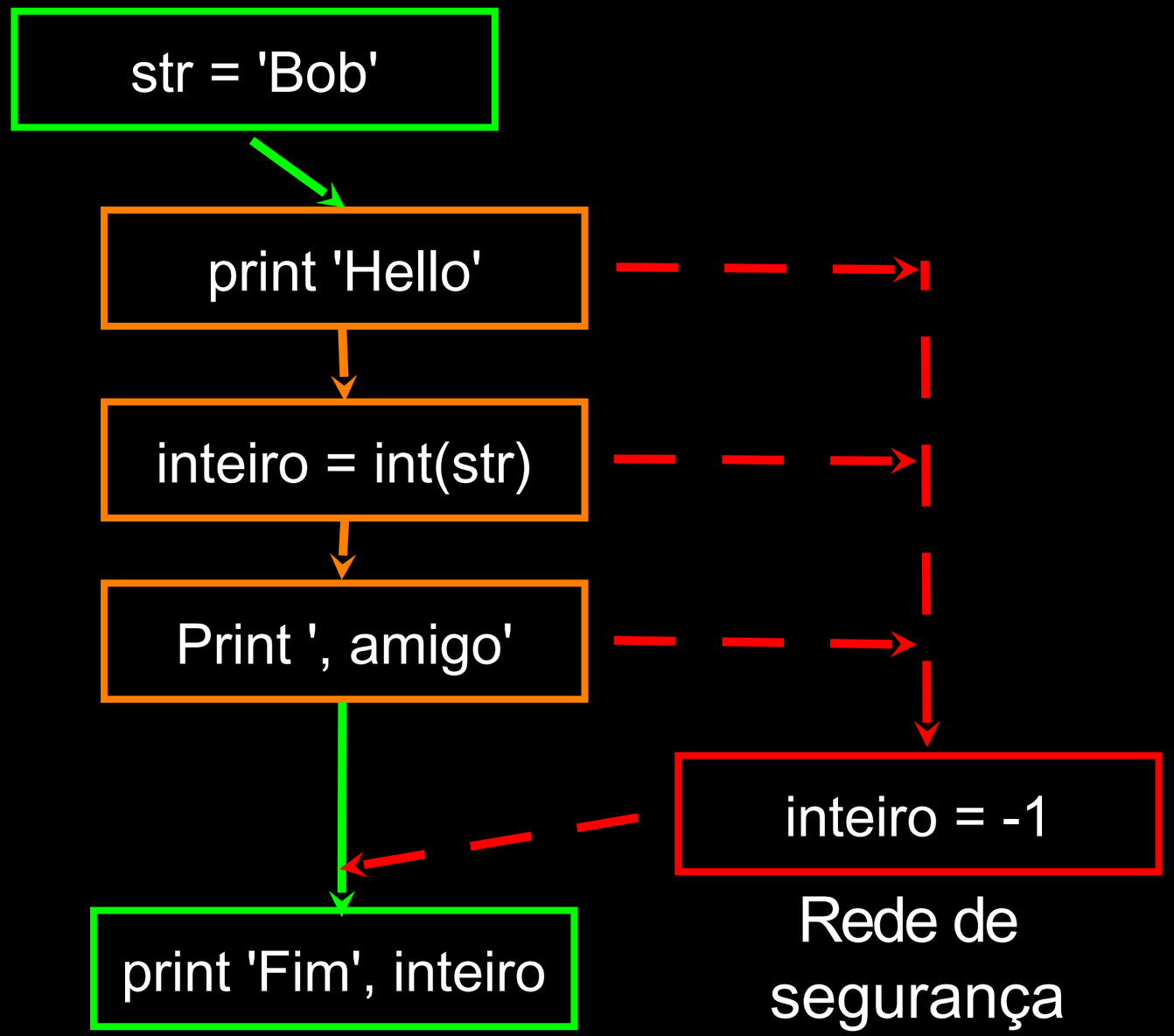
Quando a primeira conversão falha – pula-se para dentro da cláusula except: e o programa continua

```
$ python tryexcept.py
Primeiro -1
Segundo 123
```

Quando a segunda conversão tem sucesso - pula-se a cláusula except: e o programa continua

try / except

```
str = 'Bob'
try:
    print 'Ola'
    inteiro = int(str)
    Print ', amigo'
except:
    inteiro = -1

print 'Fim', inteiro
```



Exemplo de try / except

```
rawstr = raw_input('Numero:')
try:
    ival = int(rawstr)
except:
    ival = -1

if ival > 0 :
    print 'Bom trabalho'
else:
    print 'Nao e numero'
```

```
$ python trynum.py
Numero:42
Bom trabalho
$ python trynum.py
Enter a number: trinta
Nao e numero
$
```

Acknowledgements / Contributions
Agradecimentos / Contribuições

...

Initial Development: Charles Severance, University of Michigan School of Information

These slides were translated and adapted by Alberto Costa Neto (albertocn.sytes.net) of the Federal University of Sergipe