MC102 — Condicionais

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2025-03-17 14:00

Testando o valor de uma variável

No terminal do Python:

```
1 >>> x = 2
```

Testando o valor de uma variável

No terminal do Python:

```
1 >>> x = 2
2 >>> x == 2
```

Testando o valor de uma variável

No terminal do Python:

```
1 >>> x = 2
2 >>> x == 2
3 True
```

Testando o valor de uma variável

No terminal do Python:

```
1 >>> x = 2
2 >>> x == 2
3 True
4 >>> x == 3
```

Testando o valor de uma variável

No terminal do Python:

```
1 >>> x = 2
2 >>> x == 2
3 True
4 >>> x == 3
5 False
```

Testando o valor de uma variável

No terminal do Python:

```
1 >>> x = 2
2 >>> x == 2
3 True
4 >>> x == 3
5 False
6 >>> type(x == 2)
```

Testando o valor de uma variável

No terminal do Python:

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> type(x == 2)
<class 'bool'>
```

Testando o valor de uma variável

No terminal do Python:

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> type(x == 2)
<class 'bool'>
>>> type(x == 3)
```

Testando o valor de uma variável

No terminal do Python:

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> type(x == 2)
<class 'bool'>
>>> type(x == 3)
<class 'bool'>
```

Testando o valor de uma variável

No terminal do Python:

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> type(x == 2)
<class 'bool'>
>>> type(x == 3)
<class 'bool'>
>>> type(True)
```

Testando o valor de uma variável

No terminal do Python:

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> type(x == 2)
<class 'bool'>
>>> type(x == 3)
<class 'bool'>
>>> type(True)
<class 'bool'>
```

Testando o valor de uma variável

No terminal do Python:

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> type(x == 2)
<class 'bool'>
>>> type(x == 3)
<class 'bool'>
>>> type(True)
<class 'bool'>
>>> type(False)
```

Testando o valor de uma variável

No terminal do Python:

```
>>> x = 2
>>> x == 2
True
>>> x == 3
False
>>> type(x == 2)
<class 'bool'>
>>> type(x == 3)
<class 'bool'>
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

O tipo bool

O tipo `bool` define duas constantes:

O tipo bool

O tipo `bool` define duas constantes:

- `True`

O tipo bool

O tipo `bool` define duas constantes:

- `True`
- `False`

O tipo bool

O tipo `bool` define duas constantes:

- `True`
- `False`

E várias operações devolvem um `bool`

Testando se um número é par

Queremos definir se um número n dado é par ou ímpar

Testando se um número é par

Queremos definir se um número n dado é par ou ímpar

- Isto é, $n = 2k$ ou $n = 2k + 1$ para k inteiro

Testando se um número é par

Queremos definir se um número n dado é par ou ímpar

- Isto é, $n = 2k$ ou $n = 2k + 1$ para k inteiro

Qual é uma boa forma de testar se n é par ou não?

Testando se um número é par

Queremos definir se um número n dado é par ou ímpar

- Isto é, $n = 2k$ ou $n = 2k + 1$ para k inteiro

Qual é uma boa forma de testar se n é par ou não?

- Se n for par, então $n \,\%\, 2$ é 0

Testando se um número é par

Queremos definir se um número n dado é par ou ímpar

- Isto é, $n = 2k$ ou $n = 2k + 1$ para k inteiro

Qual é uma boa forma de testar se n é par ou não?

- Se n for par, então $n \,\%\, 2$ é 0
- Se n for ímpar, então $n \,\%\, 2$ é 1

Pseudocódigo

Antes do Python, vamos pensar abstratamente

Pseudocódigo

Antes do Python, vamos pensar abstratamente

- O que precisa ser feito?

Pseudocódigo

Antes do Python, vamos pensar abstratamente

- O que precisa ser feito?

```
1 Leia n
2 Se n for par
3     Imprima "n é par"
4 Senão
5     Imprima "n é impar"
```

Pseudocódigo

Porém, precisamos ter cuidado para que

Pseudocódigo

Porém, precisamos ter cuidado para que

- Cada passo seja suficientemente simples

Pseudocódigo

Porém, precisamos ter cuidado para que

- Cada passo seja suficientemente simples
- E que possa ser executado pelo computador

Pseudocódigo

Porém, precisamos ter cuidado para que

- Cada passo seja suficientemente simples
- E que possa ser executado pelo computador

Um pseudocódigo mais claro seria:

Pseudocódigo

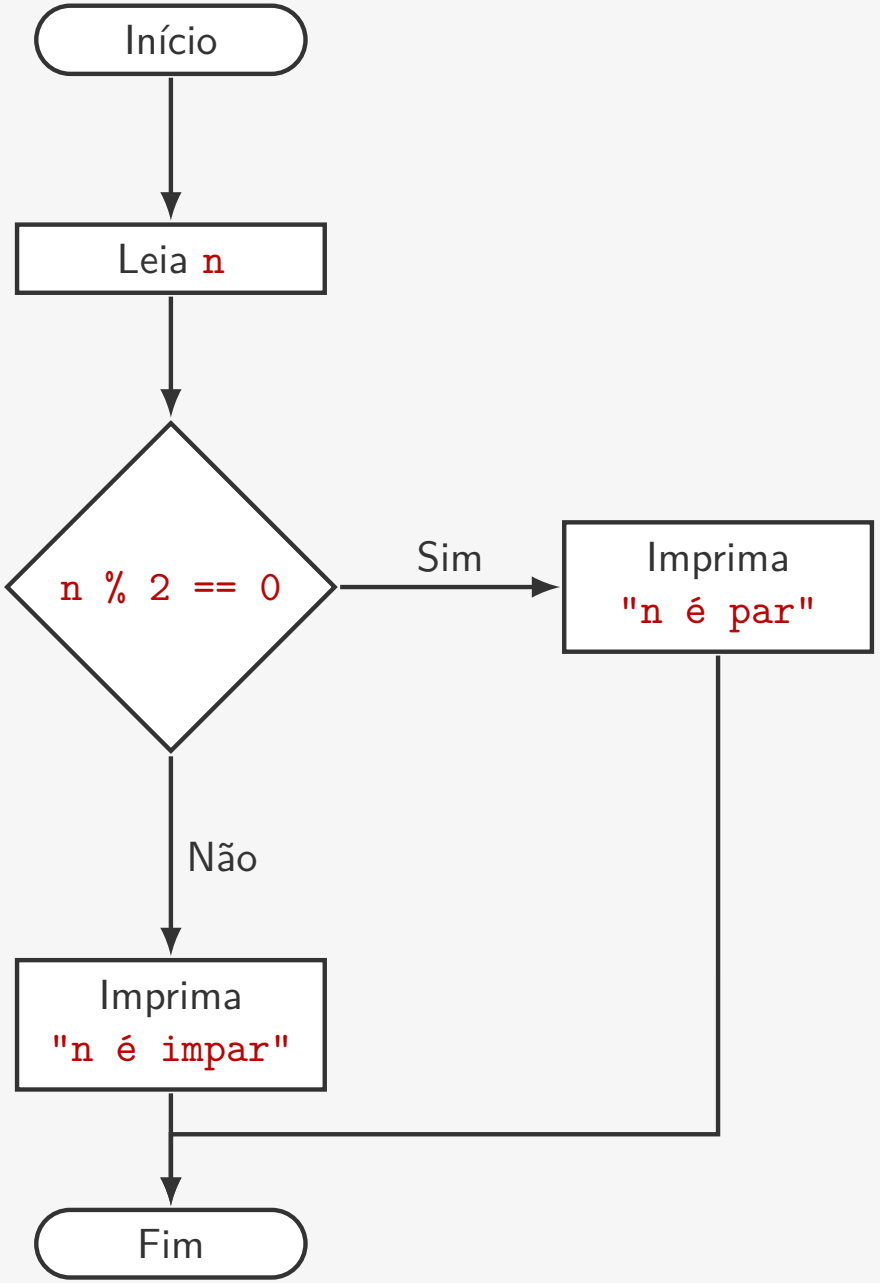
Porém, precisamos ter cuidado para que

- Cada passo seja suficientemente simples
- E que possa ser executado pelo computador

Um pseudocódigo mais claro seria:

```
1 Leia n
2 Se n % 2 == 0
3     Imprima "n é par"
4 Senão
5     Imprima "n é impar"
```

Fluxograma



Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

- Ela não é opcional como em outras linguagens

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

- Ela não é opcional como em outras linguagens
- E precisa ser consistente

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

- Ela não é opcional como em outras linguagens
- E precisa ser consistente
  - Quatro espaços é o recomendado

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

- Ela não é opcional como em outras linguagens
- E precisa ser consistente
  - Quatro espaços é o recomendado

O `:` indica que a linha do `if`/`else` terminou

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

- Ela não é opcional como em outras linguagens
- E precisa ser consistente
  - Quatro espaços é o recomendado

O `:` indica que a linha do `if`/`else` terminou

- E que um bloco de código irá começar

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

- Ela não é opcional como em outras linguagens
- E precisa ser consistente
  - Quatro espaços é o recomendado

O `:` indica que a linha do `if`/`else` terminou

- E que um bloco de código irá começar
- Ele é usado em vários outros comandos também

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

O Python utliza a indentação para criar blocos de código

- Ela não é opcional como em outras linguagens
- E precisa ser consistente
  - Quatro espaços é o recomendado

O `:` indica que a linha do `if`/`else` terminou

- E que um bloco de código irá começar
- Ele é usado em vários outros comandos também

O `if ... else`:

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

- O Python utliza a indentação para criar blocos de código
  - Ela não é opcional como em outras linguagens
  - E precisa ser consistente
    - Quatro espaços é o recomendado
- O `:` indica que a linha do `if`/`else` terminou
  - E que um bloco de código irá começar
  - Ele é usado em vários outros comandos também
- O `if ... else`:
  - Verifica o valor da expressão booleana

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

- O Python utliza a indentação para criar blocos de código
  - Ela não é opcional como em outras linguagens
  - E precisa ser consistente
    - Quatro espaços é o recomendado
- O `:` indica que a linha do `if`/`else` terminou
  - E que um bloco de código irá começar
  - Ele é usado em vários outros comandos também
- O `if ... else`:
  - Verifica o valor da expressão booleana
  - Se for `True`, executa o bloco de código do `if`

Código em Python

```
1 n = int(input("Entre com n: "))
2
3 if n % 2 == 0:
4     print(n, "é par")
5 else:
6     print(n, "é impar")
```

- O Python utliza a indentação para criar blocos de código
  - Ela não é opcional como em outras linguagens
  - E precisa ser consistente
    - Quatro espaços é o recomendado
- O `:` indica que a linha do `if`/`else` terminou
  - E que um bloco de código irá começar
  - Ele é usado em vários outros comandos também
- O `if ... else`:
  - Verifica o valor da expressão booleana
  - Se for `True`, executa o bloco de código do `if`
  - Se for `False`, executa o bloco de código do `else`

Outras comparações

Além de igualdade (==), podemos usar também:

Outras comparações

Além de igualdade (==), podemos usar também:

- < para saber se $a < b$

Outras comparações

Além de igualdade (==), podemos usar também:

- < para saber se $a < b$
- > para saber se $a > b$

Outras comparações

Além de igualdade (==), podemos usar também:

- < para saber se $a < b$
- > para saber se $a > b$
- <= para saber se $a \leq b$

Outras comparações

Além de igualdade (==), podemos usar também:

- < para saber se $a < b$
- > para saber se $a > b$
- <= para saber se $a \leq b$
- >= para saber se $a \geq b$

Outras comparações

Além de igualdade (`==`), podemos usar também:

- `<` para saber se $a < b$
- `>` para saber se $a > b$
- `<=` para saber se $a \leq b$
- `>=` para saber se $a \geq b$
- `!=` para saber se $a \neq b$

Outras comparações

Além de igualdade (==), podemos usar também:

- < para saber se $a < b$
- > para saber se $a > b$
- <= para saber se $a \leq b$
- >= para saber se $a \geq b$
- != para saber se $a \neq b$

Juntamente com o ==, são chamados de operadores de comparação

Exercício

Um inteiro n é divisível por um inteiro q se existe um inteiro a tal que $n = aq$

Exercício

Um inteiro n é divisível por um inteiro q se existe um inteiro a tal que $n = aq$

- Isto é, se $n \% q = 0$

Exercício

Um inteiro n é divisível por um inteiro q se existe um inteiro a tal que $n = aq$

- Isto é, se $n \,\%\, q = 0$

Queremos verificar se `n` é divisível por 2 ou por 3

Uma solução

Queremos verificar se `n` é divisível por `2` ou por `3`

Uma solução

Queremos verificar se `n` é divisível por `2` ou por `3`

- Podemos usar dois `if`s em sequência

Uma solução

Queremos verificar se `n` é divisível por `2` ou por `3`

- Podemos usar dois `if`s em sequência

```
1  n = int(input())
2
3  # n ser divisível por 2 é o mesmo que
4  # o resto da divisão por 2 ser 0
5  if n % 2 == 0:
6      print(n, "é divisível por 2")
7  else:
8      print(n, "não é divisível por 2")
9  if n % 3 == 0:
10     print(n, "é divisível por 3")
11 else:
12     print(n, "não é divisível por 3")
```

Uma solução

Queremos verificar se `n` é divisível por `2` ou por `3`

- Podemos usar dois `if`s em sequência

```
1  n = int(input())
2
3  # n ser divisível por 2 é o mesmo que
4  # o resto da divisão por 2 ser 0
5  if n % 2 == 0:
6      print(n, "é divisível por 2")
7  else:
8      print(n, "não é divisível por 2")
9  if n % 3 == 0:
10     print(n, "é divisível por 3")
11 else:
12     print(n, "não é divisível por 3")
```

As linhas 3 e 4 são comentários:

Uma solução

Queremos verificar se `n` é divisível por `2` ou por `3`

- Podemos usar dois `if`s em sequência

```
1  n = int(input())
2
3  # n ser divisível por 2 é o mesmo que
4  # o resto da divisão por 2 ser 0
5  if n % 2 == 0:
6      print(n, "é divisível por 2")
7  else:
8      print(n, "não é divisível por 2")
9  if n % 3 == 0:
10     print(n, "é divisível por 3")
11 else:
12     print(n, "não é divisível por 3")
```

As linhas 3 e 4 são comentários:

- Servem para entender melhor o código

Uma solução

Queremos verificar se `n` é divisível por 2 ou por 3

- Podemos usar dois `if`s em sequência

```
1  n = int(input())
2
3  # n ser divisível por 2 é o mesmo que
4  # o resto da divisão por 2 ser 0
5  if n % 2 == 0:
6      print(n, "é divisível por 2")
7  else:
8      print(n, "não é divisível por 2")
9  if n % 3 == 0:
10     print(n, "é divisível por 3")
11 else:
12     print(n, "não é divisível por 3")
```

As linhas 3 e 4 são comentários:

- Servem para entender melhor o código
- São ignoradas pelo Python

Uma solução

Queremos verificar se `n` é divisível por 2 ou por 3

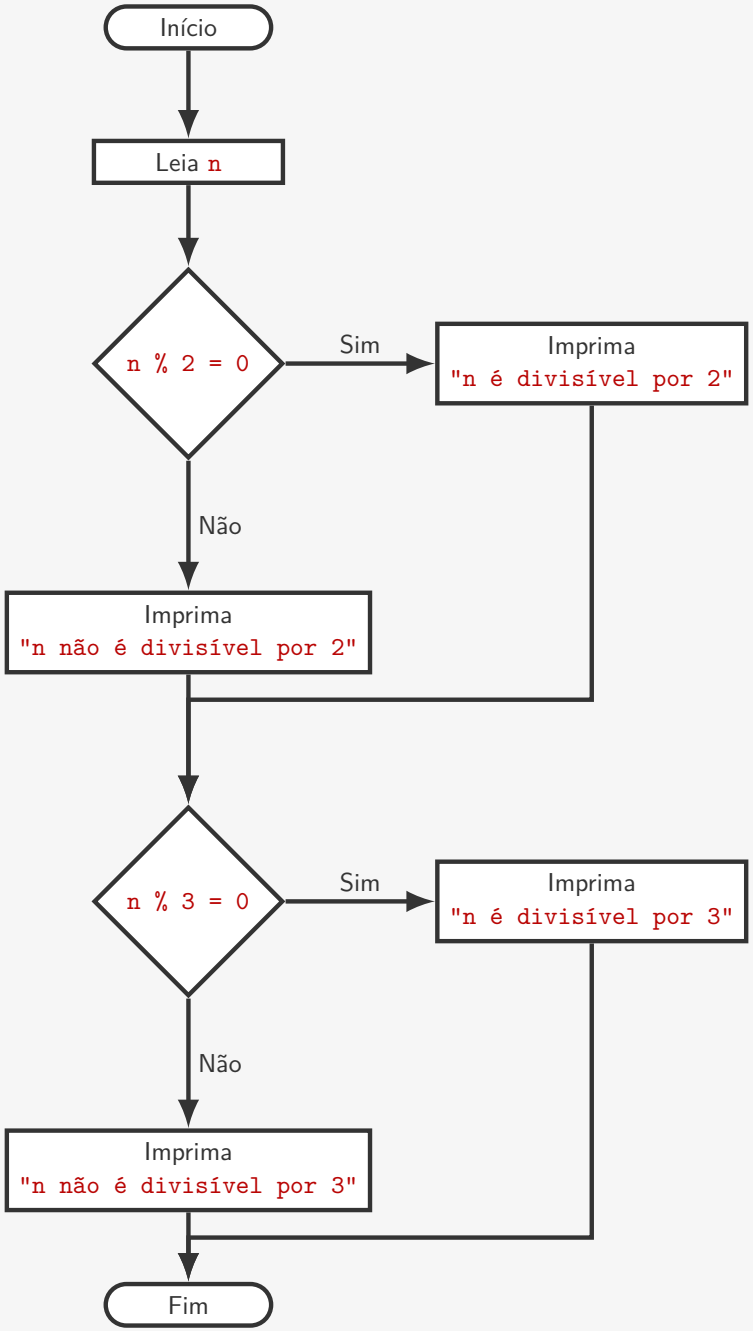
- Podemos usar dois `if`s em sequência

```
1  n = int(input())
2
3  # n ser divisível por 2 é o mesmo que
4  # o resto da divisão por 2 ser 0
5  if n % 2 == 0:
6      print(n, "é divisível por 2")
7  else:
8      print(n, "não é divisível por 2")
9  if n % 3 == 0:
10     print(n, "é divisível por 3")
11 else:
12     print(n, "não é divisível por 3")
```

As linhas 3 e 4 são comentários:

- Servem para entender melhor o código
- São ignoradas pelo Python
- Comentários devem ser usados, mas com moderação

Fluxograma


Início
Leia n
n % 2 = 0
Sim
Imprima
"n é divisível por 2"
Não
Imprima
"n não é divisível por 2"
n % 3 = 0
Sim
Imprima
"n é divisível por 3"
Não
Imprima
"n não é divisível por 3"
Fim

Exercício

Queremos verificar se n

Exercício

Queremos verificar se n

- é divisível por 2

Exercício

Queremos verificar se n

- é divisível por 2
- e não é divisível por 3

Uma solução

Queremos verificar se n

Uma solução

Queremos verificar se n

- é divisível por 2

Uma solução

Queremos verificar se n

- é divisível por 2
- e não é divisível por 3

Uma solução

Queremos verificar se n

- é divisível por 2
- e não é divisível por 3

Podemos usar um `if` dentro do outro!

Uma solução

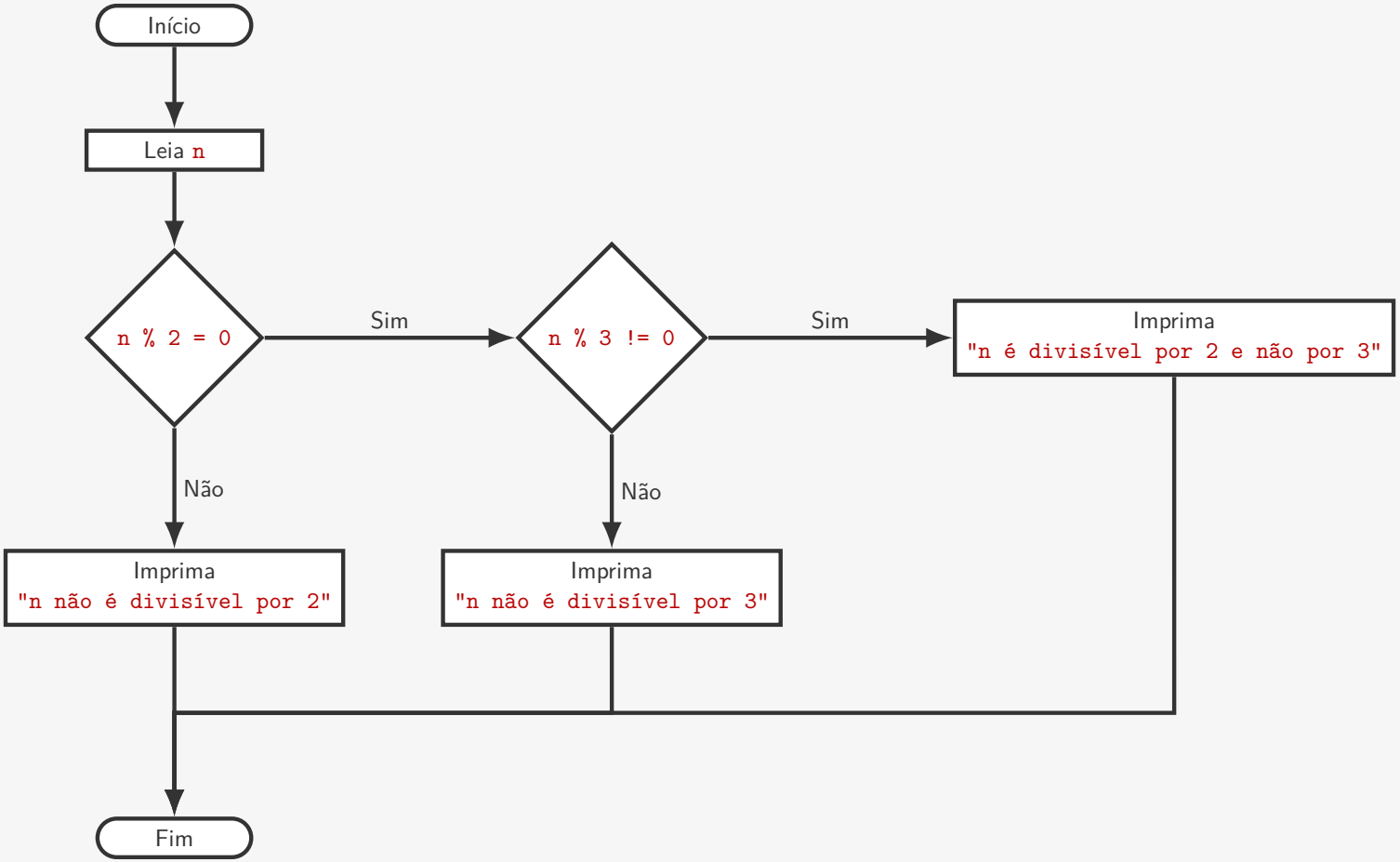
Queremos verificar se n

- é divisível por 2
- e não é divisível por 3

Podemos usar um `if` dentro do outro!

```
1 n = int(input())
2
3 if n % 2 == 0:
4     if n % 3 != 0:
5         print(n, "é divisível por 2 e não por 3")
6     else:
7         print(n, "é divisível por 2 e por 3")
8 else:
9     print(n, "não é divisível por 2")
```

Fluxograma


Início
Leia n
n % 2 = 0
Sim
n % 3 != 0
Sim
Imprima
"n é divisível por 2 e não por 3"
Não
Não
Imprima
"n não é divisível por 2"
Imprima
"n não é divisível por 3"
Fim

Exercícios

1. Escreva um programa que lê dois números e encontra o maior dos dois.

2. Escreva um programa que lê dois números inteiros x e y, sendo que y tem apenas um dígito (na base 10) e verifica se y é o último dígito (na base 10) de x.

3. Escreva um programa que lê dois números inteiros x e y e diz qual quadrante do espaço (x, y) está.

Exercício

O tempo Unix nos diz quantos segundos se passaram desde a Época Unix (00:00 de 01 de Janeiro de 1970 — UTC).

Exemplo:

- Se o tempo Unix atual é 3600, então estamos em 01:00 de 01/01/1970 (UTC)
- Se o tempo Unix é 86400, então estamos em 00:00 de 02/01/1970 (UTC).

Escreva um programa que dado um tempo Unix diz qual é o dia da semana daquele tempo.

Continua na próxima aula!

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Observações:

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Observações:

- `a` e `b` podem ser quaisquer expressões booleanas

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Observações:

- `a` e `b` podem ser quaisquer expressões booleanas
- Podemos escrever expressões do tipo `a and not b or c`

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Observações:

- `a` e `b` podem ser quaisquer expressões booleanas
- Podemos escrever expressões do tipo `a and not b or c`
- `not` precede `and` que precede `or`

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Observações:

- `a` e `b` podem ser quaisquer expressões booleanas
- Podemos escrever expressões do tipo `a and not b or c`
- `not` precede `and` que precede `or`
- Mais fácil usar parênteses do que lembrar...

Operadores lógicos

O Python tem três operadores lógicos: `and`, `or` e `not`

a	b	a and b	a or b	not a
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Observações:

- `a` e `b` podem ser quaisquer expressões booleanas
- Podemos escrever expressões do tipo `a and not b or c`
- `not` precede `and` que precede `or`
- Mais fácil usar parênteses do que lembrar...
  - `(a and (not b)) or c`

Exemplo

```
1 n = int(input())
2
3 if n % 2 == 0 or n % 3 == 0:
4     print(n, "é divisível por 2 ou por 3")
5 else:
6     print(n, "não é divisível por 2 e", end=" ")
7     print("não é divisível por 3")
```

Três soluções de um problema

Determinar se n não é divisível por 3

Três soluções de um problema

Determinar se n não é divisível por 3

Solução 1:

```
1 n = int(input())
2 if n % 3 != 0:
3     print(n, "não é divisível por 3")
4 else:
5     print(n, "é divisível por 3")
```

Três soluções de um problema

Determinar se n não é divisível por 3

Solução 1:

```
1 n = int(input())
2 if n % 3 != 0:
3     print(n, "não é divisível por 3")
4 else:
5     print(n, "é divisível por 3")
```

Solução 2:

```
1 n = int(input())
2 if not n % 3 == 0:
3     print(n, "não é divisível por 3")
4 else:
5     print(n, "é divisível por 3")
```

Três soluções de um problema

Determinar se n não é divisível por 3

Solução 1:

```
1 n = int(input())
2 if n % 3 != 0:
3     print(n, "não é divisível por 3")
4 else:
5     print(n, "é divisível por 3")
```

Solução 2:

```
1 n = int(input())
2 if not n % 3 == 0:
3     print(n, "não é divisível por 3")
4 else:
5     print(n, "é divisível por 3")
```

Solução 3:

```
1 n = int(input())
2 if n % 3 == 1 or n % 3 == 2:
3     print(n, "não é divisível por 3")
4 else:
5     print(n, "é divisível por 3")
```

Regras de De Morgan

Uma regra útil da lógica:

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`
- `not (a and b)` é equivalente a `(not a) or (not b)`

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`
- `not (a and b)` é equivalente a `(not a) or (not b)`

Isso permite

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`
- `not (a and b)` é equivalente a `(not a) or (not b)`

Isso permite

- Escrever expressões menores ou mais legíveis

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`
- `not (a and b)` é equivalente a `(not a) or (not b)`

Isso permite

- Escrever expressões menores ou mais legíveis
- Ou saber porque uma condição falhou

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`
- `not (a and b)` é equivalente a `(not a) or (not b)`

Isso permite

- Escrever expressões menores ou mais legíveis
- Ou saber porque uma condição falhou
  - Se o `if a and b` falhou é porque

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`
- `not (a and b)` é equivalente a `(not a) or (not b)`

Isso permite

- Escrever expressões menores ou mais legíveis
- Ou saber porque uma condição falhou
  - Se o `if a and b` falhou é porque
  - `not a` ou `not b` é verdade

Regras de De Morgan

Uma regra útil da lógica:

- `not (a or b)` é equivalente a `(not a) and (not b)`
- `not (a and b)` é equivalente a `(not a) or (not b)`

Isso permite

- Escrever expressões menores ou mais legíveis
- Ou saber porque uma condição falhou
  - Se o `if a and b` falhou é porque
  - `not a` ou `not b` é verdade
  - Isto é, ou `a` ou `b` é falso (ou os dois)

Voltando a um exercício anterior

```
1 n = int(input())
2
3 if n % 2 == 0 and (not n % 3 == 0):
4     print(n, "é divisível por 2 e não por 3")
5 if not n % 2 == 0:
6     print(n, "não é divisível por 2")
7 if n % 3 == 0:
8     print(n, "é divisível por 3")
```

Voltando a um exercício anterior

```
1 n = int(input())
2
3 if n % 2 == 0 and (not n % 3 == 0):
4     print(n, "é divisível por 2 e não por 3")
5 if not n % 2 == 0:
6     print(n, "não é divisível por 2")
7 if n % 3 == 0:
8     print(n, "é divisível por 3")
```

Usamos De Morgan para verificar porque não executa a linha 4

Voltando a um exercício anterior

```
1 n = int(input())
2
3 if n % 2 == 0 and (not n % 3 == 0):
4     print(n, "é divisível por 2 e não por 3")
5 if not n % 2 == 0:
6     print(n, "não é divisível por 2")
7 if n % 3 == 0:
8     print(n, "é divisível por 3")
```

Usamos De Morgan para verificar porque não executa a linha 4

Note que um `if` não precisa ter um `else`!

Voltando a um exercício anterior

```
1 n = int(input())
2
3 if n % 2 == 0 and (not n % 3 == 0):
4     print(n, "é divisível por 2 e não por 3")
5 if not n % 2 == 0:
6     print(n, "não é divisível por 2")
7 if n % 3 == 0:
8     print(n, "é divisível por 3")
```

Usamos De Morgan para verificar porque não executa a linha 4

Note que um `if` não precisa ter um `else`!

- Mas todo `else` precisa ter um `if`...

Voltando a um exercício anterior

```
1 n = int(input())
2
3 if n % 2 == 0 and (not n % 3 == 0):
4     print(n, "é divisível por 2 e não por 3")
5 if not n % 2 == 0:
6     print(n, "não é divisível por 2")
7 if n % 3 == 0:
8     print(n, "é divisível por 3")
```

Usamos De Morgan para verificar porque não executa a linha 4

Note que um `if` não precisa ter um `else`!

- Mas todo `else` precisa ter um `if`...

O que é impresso se `n` for `3`?

Voltando a um exercício anterior

```
1 n = int(input())
2
3 if n % 2 == 0 and (not n % 3 == 0):
4     print(n, "é divisível por 2 e não por 3")
5 if not n % 2 == 0:
6     print(n, "não é divisível por 2")
7 if n % 3 == 0:
8     print(n, "é divisível por 3")
```

Usamos De Morgan para verificar porque não executa a linha 4

Note que um `if` não precisa ter um `else`!

- Mas todo `else` precisa ter um `if`...

O que é impresso se `n` for `3`?

- Isso não acontecia no programa anterior...

Uma versão melhor

```
1  n = int(input())
2
3  if n % 2 == 0 and (not n % 3 == 0):
4      print(n, "é divisível por 2 e não por 3")
5  elif n % 2 == 0:
6      print(n, "é divisível por 2 e por 3")
7  elif not n % 3 == 0:
8      print(n, "não é divisível por 2 e por 3")
9  else:
10     print(n, "não é divisível por 2 e é por 3")
```

Uma versão melhor

```
1  n = int(input())
2
3  if n % 2 == 0 and (not n % 3 == 0):
4      print(n, "é divisível por 2 e não por 3")
5  elif n % 2 == 0:
6      print(n, "é divisível por 2 e por 3")
7  elif not n % 3 == 0:
8      print(n, "não é divisível por 2 e por 3")
9  else:
10     print(n, "não é divisível por 2 e é por 3")
```

O `elif` (de *else if*) testa uma nova condição

Uma versão melhor

```
1  n = int(input())
2
3  if n % 2 == 0 and (not n % 3 == 0):
4      print(n, "é divisível por 2 e não por 3")
5  elif n % 2 == 0:
6      print(n, "é divisível por 2 e por 3")
7  elif not n % 3 == 0:
8      print(n, "não é divisível por 2 e por 3")
9  else:
10     print(n, "não é divisível por 2 e é por 3")
```

O `elif` (de *else if*) testa uma nova condição

- Apenas se o `if` e os `elif`s anteriores falharam

Uma versão melhor

```
1  n = int(input())
2
3  if n % 2 == 0 and (not n % 3 == 0):
4      print(n, "é divisível por 2 e não por 3")
5  elif n % 2 == 0:
6      print(n, "é divisível por 2 e por 3")
7  elif not n % 3 == 0:
8      print(n, "não é divisível por 2 e por 3")
9  else:
10     print(n, "não é divisível por 2 e é por 3")
```

O `elif` (de *else if*) testa uma nova condição

- Apenas se o `if` e os `elif`s anteriores falharam

Exercício: como fazer sem usar `elif`?

Exercício

Vamos fazer um programa em Python que:

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c
- e calcula as raízes de $ax^2 + bx + c = 0$

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c
- e calcula as raízes de $ax^2 + bx + c = 0$
- usando a fórmula de Bhaskara

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c
- e calcula as raízes de $ax^2 + bx + c = 0$
- usando a fórmula de Bhaskara

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c
- e calcula as raízes de $ax^2 + bx + c = 0$
- usando a fórmula de Bhaskara

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Cuidados:

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c
- e calcula as raízes de $ax^2 + bx + c = 0$
- usando a fórmula de Bhaskara

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Cuidados:

- a não pode ser zero!

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c
- e calcula as raízes de $ax^2 + bx + c = 0$
- usando a fórmula de Bhaskara

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Cuidados:

- a não pode ser zero!
- $b^2 - 4ac$ não pode ser negativo!

Exercício

Vamos fazer um programa em Python que:

- lê três números a, b e c
- e calcula as raízes de $ax^2 + bx + c = 0$
- usando a fórmula de Bhaskara

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Cuidados:

- a não pode ser zero!
- $b^2 - 4ac$ não pode ser negativo!

Vamos resolver no VS Code!

Solução

```
1  # equação a x**2 + b x + c = 0
2  a = float(input())
3  b = float(input())
4  c = float(input())
5
6  delta = b**2 - 4 * a * c
7
8  if (a == 0):
9      # Expressão da forma b x + c = 0
10     print("Não é uma equação de segundo grau!")
11 elif (delta < 0):
12     print("As raízes são números complexos!")
13 else:
14     print("Raizes:")
15     print((- b - delta**(1 / 2)) / (2 * a))
16     print((- b + delta**(1 / 2)) / (2 * a))
```

Exercícios

1. Escreva um programa que lê dois números inteiros x e y e diz qual quadrante do espaço (x, y) está.
   - Mas use operadores lógicos dessa vez...

2. Escreva um programa que lê três números e encontra o maior dos três.

3. Dê um exemplo onde utilizar `if x ... else` é diferente de usar `if x` seguido de `if not x`.