# Números Primos

## Algoritmos e Programação

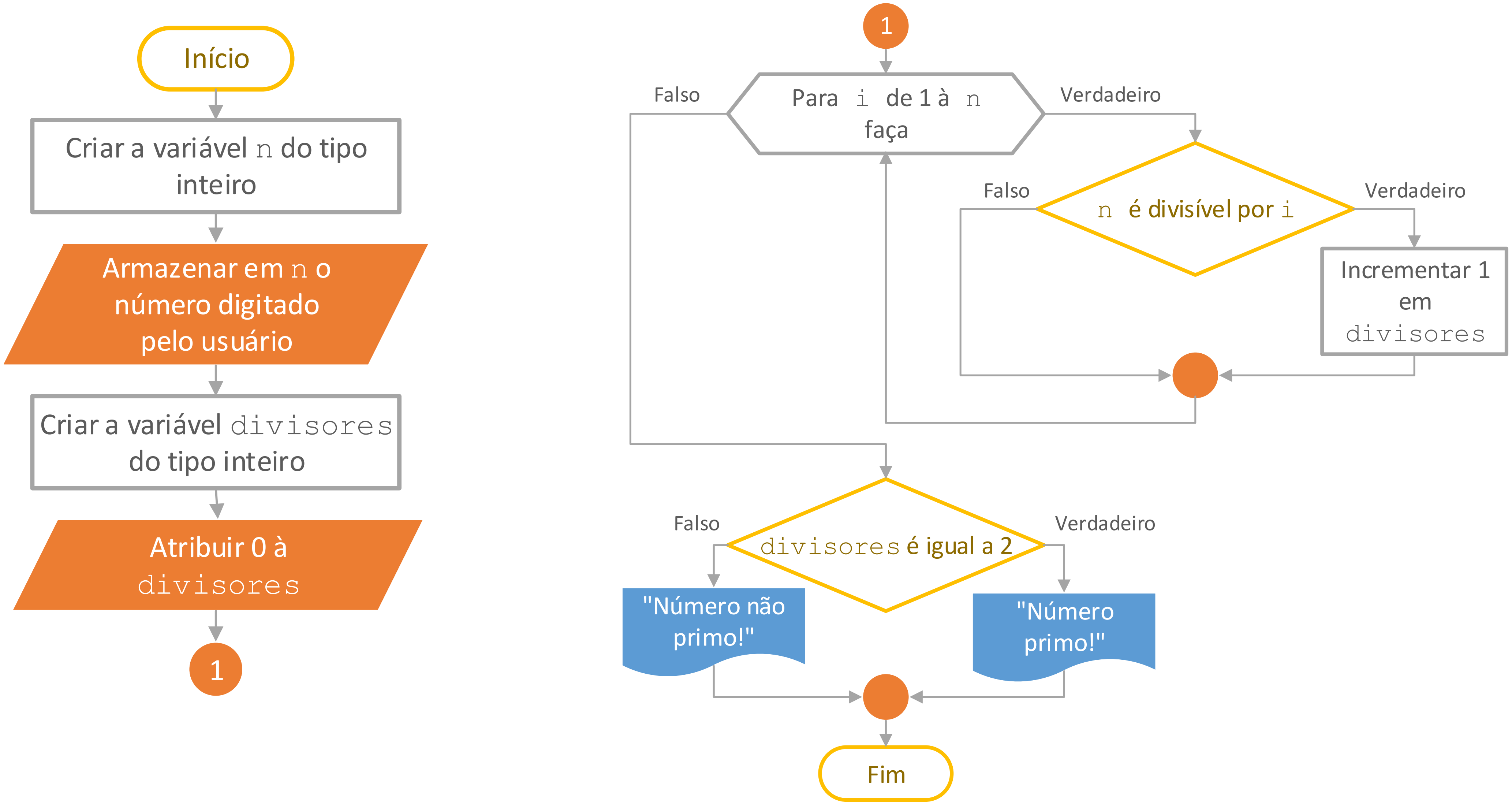
• Prof. Lucio Nunes de Lira •

# O QUE É?

Um **número natural primo** é um número natural maior do que um e que tem exatamente **dois divisores naturais**: um e ele próprio.

**Exemplos:** 2, 3, 5, 7, 11, 13, 17, 19…

# Como descobrir se um natural é primo



```python
# Conta os divisores de 1 até n.
def primo1(n):
    divisores = 0
    for i in range(1, n+1):
        if n % i == 0:
            divisores += 1
    if divisores == 2:
        return True
    else:
        return False
```

1ª VERSÃO

**FUNCIONAMENTO**

```
>>> primo1(11)

11 %  1 = 0 (divisor)
11 %  2 = 1
11 %  3 = 2
11 %  4 = 3
11 %  5 = 1
11 %  6 = 5
11 %  7 = 4
11 %  8 = 3
11 %  9 = 2
11 % 10 = 1
11 % 11 = 0 (divisor)
True
```

# HORA DO

# [ UPGRADE ]

```python
# Conta os divisores de 1 até n.
def primo2(n):
    divisores = 0
    for i in range(1, n+1):
        if n % i == 0:
            divisores += 1
    return divisores == 2
```

2ª VERSÃO

**Funcionamento**

```
>>> primo2(11)

11 %  1 = 0 (divisor)
11 %  2 = 1
11 %  3 = 2
11 %  4 = 3
11 %  5 = 1
11 %  6 = 5
11 %  7 = 4
11 %  8 = 3
11 %  9 = 2
11 % 10 = 1
11 % 11 = 0 (divisor)
True
```

# HORA DO

# [ UPGRADE ]

```python
# Conta os divisores de 2 até n-1.
def primo3(n):
    divisores = 0
    for i in range(2, n):
        if n % i == 0:
            divisores += 1
    return divisores == 0
```

3ª VERSÃO

**Funcionamento**

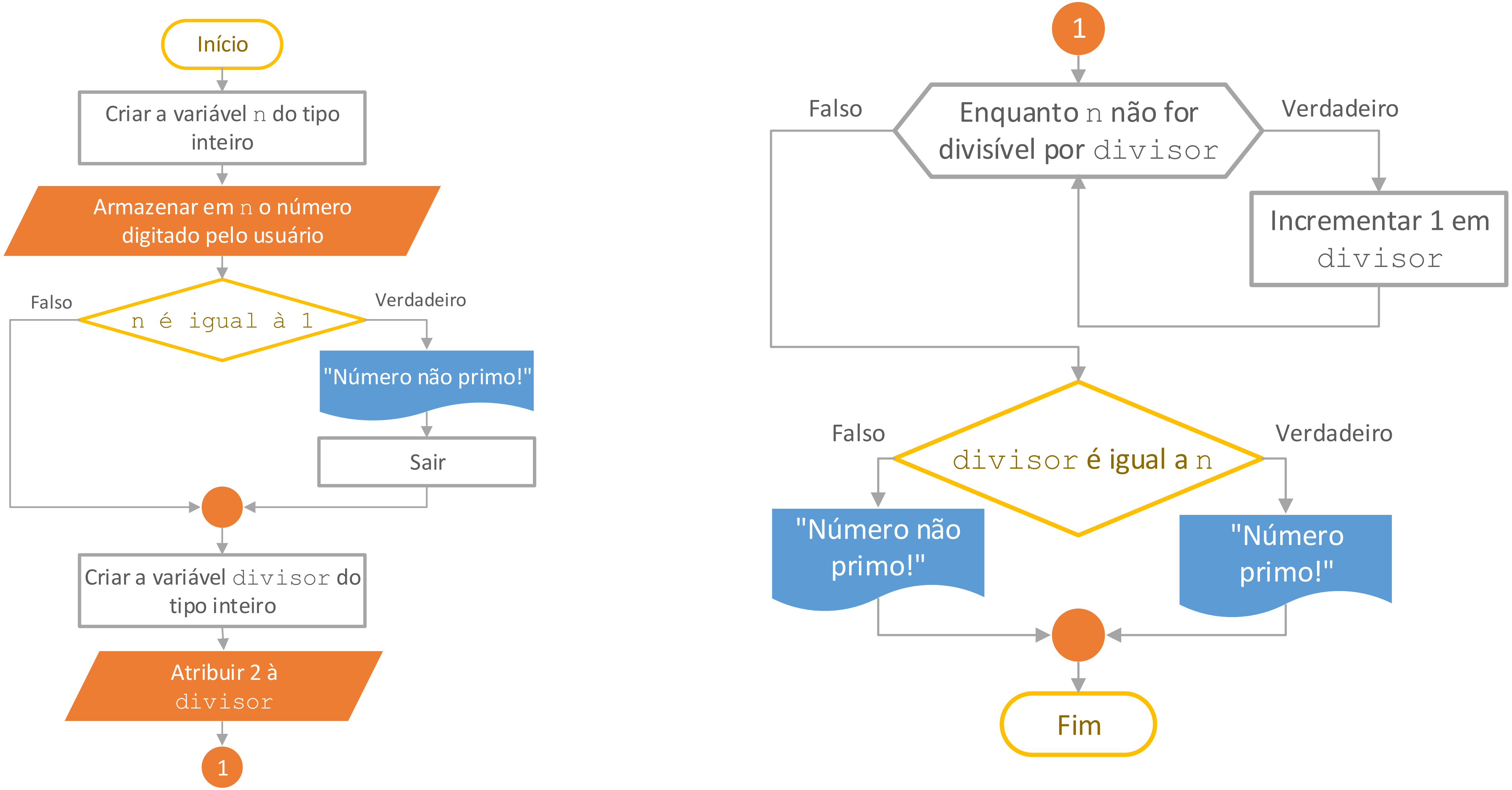
```
>>> primo3(11)

11 %  2 = 1
11 %  3 = 2
11 %  4 = 3
11 %  5 = 1
11 %  6 = 5
11 %  7 = 4
11 %  8 = 3
11 %  9 = 2
11 % 10 = 1
True
```

# HORA DO

# [ UPGRADE ]

# ALGORITMO OTIMIZADO



```python
# Só testa até o primeiro divisor.
def primo4(n):
    for divisor in range(2, n):
        if n % divisor == 0:
            return False
    return True
```

4ª VERSÃO

### Funcionamento

```
>>> primo4(49)

49 % 2 = 1
49 % 3 = 1
49 % 4 = 1
49 % 5 = 4
49 % 6 = 1
49 % 7 = 0 (divisor)
False
```

# HORA DO

# [ UPGRADE ]

5ª VERSÃO

```python
# Após 2, só testa divisores ímpares.
def primo5(n):
    if n % 2 == 0: return n == 2
    for divisor in range(3, n, 2):
        if n % divisor == 0:
            return False
    return True
```

**FUNCIONAMENTO**

```
>>> primo5(11)

11 % 2 = 1
11 % 3 = 2
11 % 5 = 1
11 % 7 = 4
11 % 9 = 2
True
```

# HORA DO

# [ UPGRADE ]

# Vamos pensar...

- Sejam $m$ e $n$ dois números naturais positivos e $m \neq n$. Então, se $m > \frac{n}{2}$, $m$ não é divisor de $n$.
- **Direto ao ponto**: não existe divisor de $n$ maior que sua metade, exceto o próprio $n$.
- Logo, é inútil buscar divisores de um número natural acima de sua metade.

## Exemplos do conceito

```
10 %  1 = 0 (divisor)
10 %  2 = 0 (divisor)
10 %  3 = 1
10 %  4 = 2
10 %  5 = 0 (divisor)
10 %  6 = 4
10 %  7 = 3
10 %  8 = 2
10 %  9 = 1
10 % 10 = 0 (divisor)
```

```
7 % 1 = 0 (divisor)
7 % 2 = 1
7 % 3 = 1
7 % 4 = 3
7 % 5 = 2
7 % 6 = 1
7 % 7 = 0 (divisor)
```

```python
# Só testa divisores até a metade de n.
def primo6(n):
    if n % 2 == 0: return n == 2
    metade = n // 2
    for divisor in range(3, metade+1, 2):
        if n % divisor == 0:
            return False
    return True
```

6ª VERSÃO

**FUNCIONAMENTO**

```
>>> primo6(11)

11 % 2 = 1
11 % 3 = 2
11 % 5 = 1
True
```

# HORA DO

# [ UPGRADE ]

- Um número natural maior que 1 é **composto** quando tem mais de dois divisores naturais **distintos**.

- Todo número composto pode ser **decomposto** em um produto de **no mínimo** dois fatores **primos** (não necessariamente distintos).

- Exemplos: **21 = 3×7** | **27 = 3×3×3** | **30 = 2×3×5** | **100 = 2×2×5×5**.

- https://www.mathsisfun.com/numbers/prime-factorization-tool.html

Considere um número natural $n > 1$ e acompanhe o raciocínio:

$$n = \sqrt{n} \times \sqrt{n}$$

Suponha $n = a \times b$. Para que essa equação seja válida:

se $a = \sqrt{n}$, então $b = \sqrt{n}$

se $a > \sqrt{n}$, então $b < \sqrt{n}$

se $a < \sqrt{n}$, então $b > \sqrt{n}$

- **Recapitulando:** se $n$ for um número composto, poderá ser decomposto em no mínimo **dois fatores** primos.

- Se $n$ for decomposto em pelo menos dois fatores, um deles deve ser $\leq \sqrt{n}$ e, como todo fator é também um divisor, $n$ tem pelo menos um divisor $\leq \sqrt{n}$.

- Ou seja, se um número natural maior que 1 não tiver um divisor **menor ou igual à sua raiz** é porque é primo.

- O divisor deve ser um natural, então **arredondaremos a raiz**.

- Na matemática seria arredondado para baixo, na computação é melhor que seja para cima, para tentar **evitar erros de precisão** inerentes ao *float*, o que poderia ocasionar uma divisão a menos.

- Para calcular a **raiz quadrada** usaremos a função `sqrt` e para arredondar para cima temos a função `ceil`, que retornará o menor inteiro maior ou igual a raiz, isto é, a **função teto**.

7ª VERSÃO

```python
from math import ceil, sqrt

# Só testa divisores até a raiz de n.
def primo7(n):
    if n % 2 == 0: return n == 2
    raiz = ceil(sqrt(n))
    for divisor in range(3, raiz+1, 2):
        if n % divisor == 0:
            return False
    return True
```

**FUNCIONAMENTO**

```
>>> primo7(101)

101 %  2 = 1
101 %  3 = 2
101 %  5 = 1
101 %  7 = 3
101 %  9 = 2
101 % 11 = 2
True
```

# Benchmark em Python

| Função | Entrada | | | |
| --- | --- | --- | --- | --- |
| | 1.000.000[1] | 1.000.003[2] | 1.000.000.000[1] | 1.000.000.007[2] |
| Primo 1 | 0,136 | 0,127 | 121,850 | 136,509 |
| Primo 2 | 0,145 | 0,113 | 119,730 | 118,500 |
| Primo 3 | 0,116 | 0,116 | 118,350 | 119,539 |
| Primo 4 | 0,000 | 0,117 | 0,000 | 118,995 |
| Primo 5 | 0,000 | 0,056 | 0,000 | 59,588 |
| Primo 6 | 0,000 | 0,028 | 0,000 | 30,283 |
| Primo 7 | 0,000 | 0,000 | 0,000 | 0,002 |

**Tabela 1:** Média do tempo de processamento por entrada (em segundos).

[1] Número não primo.

[2] Número primo.

# Benchmark em linguagem C

| Função | Entrada | | | |
|---|---|---|---|---|
| | 1.000.000[1] | 1.000.003[2] | 1.000.000.000[1] | 1.000.000.007[2] |
| Primo 1 | 0,007 | 0,007 | 6,695 | 6,695 |
| Primo 2 | 0,007 | 0,007 | 6,694 | 6,694 |
| Primo 3 | 0,007 | 0,007 | 6,693 | 6,694 |
| Primo 4 | 0,000 | 0,007 | 0,000 | 6,717 |
| Primo 5 | 0,000 | 0,003 | 0,000 | 3,359 |
| Primo 6 | 0,000 | 0,002 | 0,000 | 1,672 |
| Primo 7 | 0,000 | 0,000 | 0,000 | 0,000 |

**Tabela 2:** Média do tempo de processamento por entrada (em segundos).

[1] Número não primo.

[2] Número primo.

# Contato

## Prof. Lucio Nunes

https://www.linkedin.com/in/lucio-nunes-de-lira/