MAC 115 — Introdução à Ciência da Computação

Aula 3

Nelson Lago

IF noturno – 2023

Previously on MAC 115...

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - O algoritmo é *abstrato* (como a planta de um prédio ou uma receita de bolo)
3. ***Implementar* o algoritmo em uma linguagem de programação**
   - Gerando um *programa* que pode ser *executado* para solucionar o problema
4. **Testar o programa**

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**
   - Gerando um resultado
3. **"Faz alguma coisa" com esse resultado**
   - Mostra para o usuário
   - **Utiliza como dado para fazer outra coisa**

Lembre-se:

Lembre-se:

O computador só faz o que você manda!

Lembre-se:

O computador só faz o que você manda!

(não o que você quer)

Lembre-se:

O computador só faz o que você manda!

(não o que você quer)

Se você esquecer um passo, ele obedece!

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um ~~*resultado*~~ *valor*)
  - Exemplo: $2 + 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 + 3 + 7$
  - Exemplo: $\frac{2+3+7}{6}$

Tipos

- **Obviamente, `"2" + "3"` não é a mesma operação que `2 + 3`**
  - Embora estejamos chamando ambas de +, o computador executa cada uma delas de maneira completamente diferente
- **Existem *tipos de dados* diferentes em python**

Tipos

```
print(type(2))
print(type(False))
print(type(2.0))
print(type("Olá"))
```

Tipos

```
print(type(2))
print(type(False))
print(type(2.0))
print(type("Olá"))
```

```
<class 'int'>
```

Tipos

```
print(type(2))
print(type(False))
print(type(2.0))
print(type("Olá"))
```

```
<class 'int'>
<class 'bool'>
```

Tipos

```
print(type(2))
print(type(False))
print(type(2.0))
print(type("Olá"))
```

```
<class 'int'>
<class 'bool'>
<class 'float'>
```

Tipos

```
print(type(2))
print(type(False))
print(type(2.0))
print(type("Olá"))
```

```
<class 'int'>
<class 'bool'>
<class 'float'>
<class 'str'>
```

Expressões lógicas

```
print(2 > 3)
print(5 > 4)
print(2 > 3 and 5 > 4)
print(2 > 3 or 5 > 4)
```

Expressões lógicas

```
print(2 > 3)
print(5 > 4)
print(2 > 3 and 5 > 4)
print(2 > 3 or 5 > 4)
```

```
False
```

Expressões lógicas

```
print(2 > 3)
print(5 > 4)
print(2 > 3 and 5 > 4)
print(2 > 3 or 5 > 4)
```

```
False
True
```

Expressões lógicas

```
print(2 > 3)
print(5 > 4)
print(2 > 3 and 5 > 4)
print(2 > 3 or 5 > 4)
```

```
False
True
False
```

Expressões lógicas

```
print(2 > 3)
print(5 > 4)
print(2 > 3 and 5 > 4)
print(2 > 3 or 5 > 4)
```

```
False
True
False
True
```

Operadores lógicos

A **and** B

	A = **True**	A = **False**
B = **True**	**True**	**False**
B = **False**	**False**	**False**

*Tabela verdade do operador **and***

Operadores lógicos

A `or` B

	A = `True`	A = `False`
B = `True`	`True`	`True`
B = `False`	`True`	`False`

Tabela verdade do operador `or`

Nomes (variáveis)

- Ao programar, preferimos pensar no problema a ser resolvido e não nas idiossincrasias do computador
- Linguagens de programação de alto nível procuram oferecer os recursos para isso
- Uma das coisas mais importantes para esse fim é utilizar *nomes*

Nomes (variáveis)

x $\leftarrow$ 5 (atribuição)

Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em python com "=":

```
x = 5
```

Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em python com "=" :

```
x = 5
```

E por que não:

```
x = x + 1
```

Nomes (variáveis)

x ← 5 (atribuição)

Há um número finito de caracteres no teclado, então fazemos atribuição em python com "=" :

```
x = 5
```

E por que não:

```
x = x + 1
```

AAAAAHHHH!!!!!!!

Nomes (variáveis)

```
print(type(5))
print(type(x))
x = 5
print(type(x))
```

Nomes (variáveis)

```
print(type(5))
print(type(x))
x = 5
print(type(x))
```

```
<class 'int'>
```

Nomes (variáveis)

```
print(type(5))
print(type(x))
x = 5
print(type(x))
```

```
<class 'int'>
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
```

Nomes (variáveis)

```
print(type(5))
print(type(x))
x = 5
print(type(x))
```

```
<class 'int'>
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'x' is not defined
<class 'int'>
```

and now for something completely different

Adeus, shell!

- **Começamos usando o *shell* do python através do IDLE**

Adeus, shell!

- Começamos usando o *shell* do python através do IDLE
- MAS...

Adeus, shell!

- Começamos usando o *shell* do python através do IDLE
- MAS...
- Em geral, *não* se usa o shell; ele serve apenas para experimentar alguns comandos básicos

Adeus, shell!

- Começamos usando o *shell* do python através do IDLE
- MAS...
- Em geral, *não* se usa o shell; ele serve apenas para experimentar alguns comandos básicos
- O que fazemos é criar um arquivo de texto puro (mas usando um nome do tipo `.py` ao invés de `.txt`) com os comandos desejados

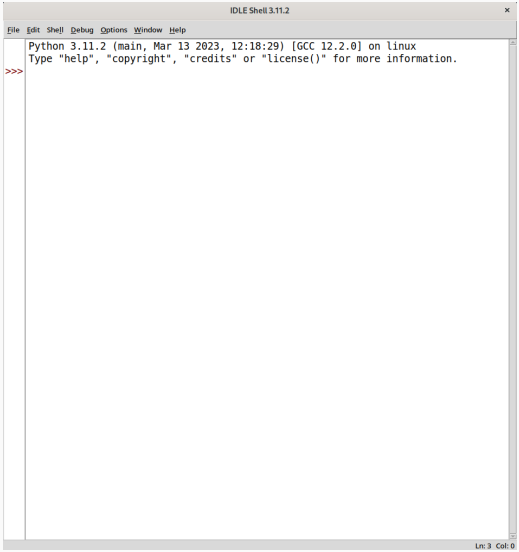
Adeus, shell!

- Começamos usando o *shell* do python através do IDLE
- MAS...
- Em geral, *não* se usa o shell; ele serve apenas para experimentar alguns comandos básicos
- O que fazemos é criar um arquivo de texto puro (mas usando um nome do tipo `.py` ao invés de `.txt`) com os comandos desejados
- Mas o que é "texto puro"?

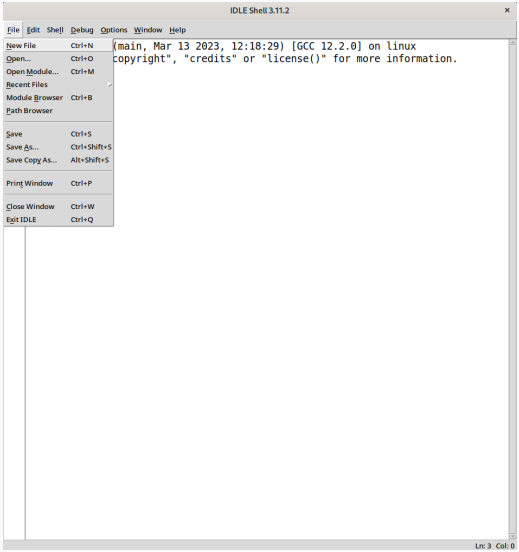
Adeus, shell!

- Começamos usando o *shell* do python através do IDLE
- MAS...
- Em geral, *não* se usa o shell; ele serve apenas para experimentar alguns comandos básicos
- O que fazemos é criar um arquivo de texto puro (mas usando um nome do tipo `.py` ao invés de `.txt`) com os comandos desejados
- Mas o que é "texto puro"?
- libreoffice vs IDLE

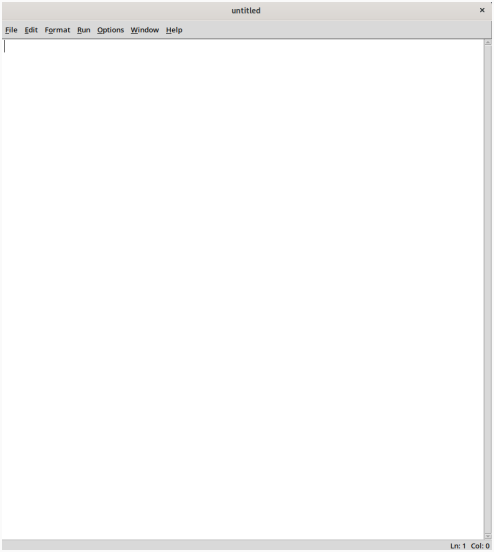
IDLE


IDLE Shell 3.11.2
File Edit Shell Debug Options Window Help
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
Ln: 3 Col: 0

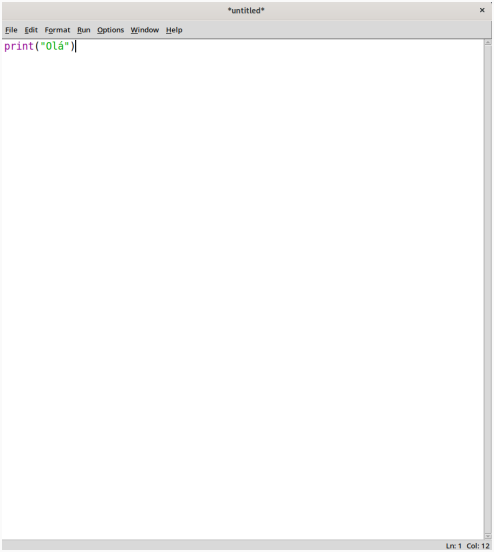
IDLE


IDLE Shell 3.11.2
File Edit Shell Debug Options Window Help
New File Ctrl+N
Open... Ctrl+O
Open Module... Ctrl+M
Recent Files
Module Browser Ctrl+B
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Window Ctrl+W
Exit IDLE Ctrl+Q
(main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
copyright", "credits" or "license()" for more information.
Ln: 3 Col: 0

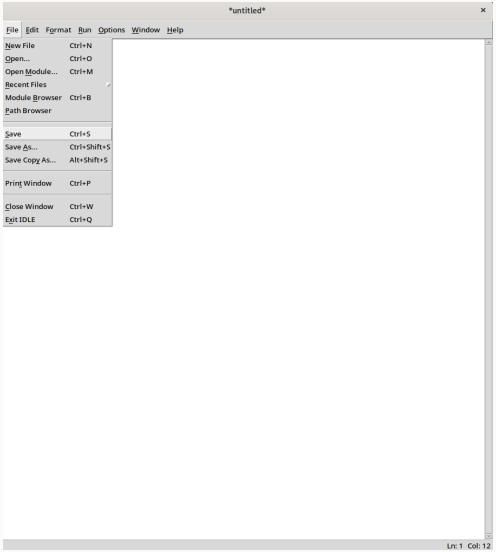
IDLE



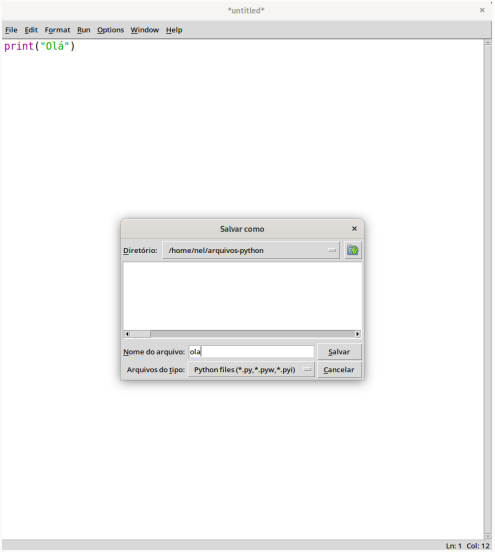
IDLE


untitled
File Edit Format Run Options Window Help
print("Olá")
Ln: 1 Col: 12

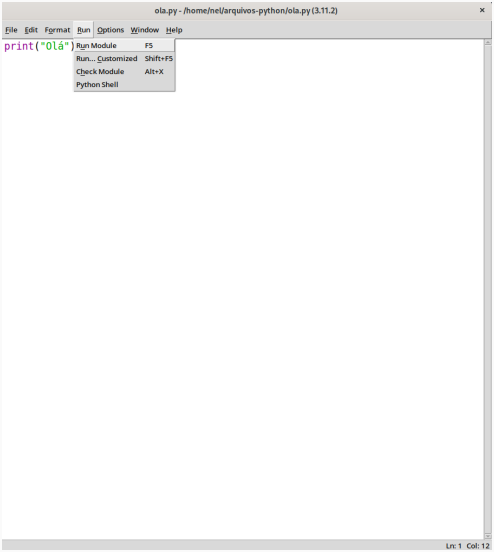
IDLE


untitled
File Edit Format Run Options Window Help
New File Ctrl+N
Open... Ctrl+O
Open Module... Ctrl+M
Recent Files
Module Browser Ctrl+B
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Window Ctrl+W
Exit IDLE Ctrl+Q
Ln: 1 Col: 12

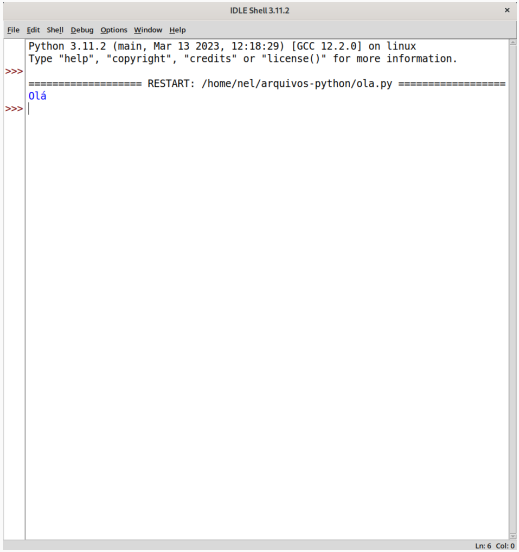
IDLE


untitled
File Edit Format Run Options Window Help
print("Olá")
Salvar como
Diretório:
/home/nel/arquivos-python
Nome do arquivo:
ola
Salvar
Arquivos do tipo:
Python files (*.py,*.pyw,*.pyi)
Cancelar
Ln: 1 Col: 12

IDLE


ola.py - /home/nel/arquivos-python/ola.py (3.11.2)
File Edit Format Run Options Window Help
print("Olá")
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell
Ln: 1 Col: 12

IDLE


IDLE Shell 3.11.2
File Edit Shell Debug Options Window Help
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: /home/nel/arquivos-python/ola.py ==========
Olá
>>>
Ln: 6 Col: 0

Nomes (variáveis)

$$-x^2 + 3x + 4 = 0$$

Nomes (variáveis)

$$-x^2 + 3x + 4 = 0$$

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

Nomes (variáveis)

$$-x^2 + 3x + 4 = 0$$

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

```
import math
print(-2 * 4/(3 + math.sqrt(3**2 - 4 * -1 * 4)),
-2 * 4/(3 - math.sqrt(3**2 - 4 * -1 * 4)))
```

```
-1.0 4.0
```

Nomes (variáveis)

$$-x^2 + 3x + 4 = 0$$

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

```
import math
print(-2 * 4/(3 + math.sqrt(3**2 - 4 * -1 * 4)),
-2 * 4/(3 - math.sqrt(3**2 - 4 * -1 * 4)))
```

```
-1.0 4.0
```

Nomes (variáveis)

```
import math
a = -1
b = 3
c = 4
delta = b**2 - 4 * a * c
raiz1 = -2 * c / (b + math.sqrt(delta))
raiz2 = -2 * c / (b - math.sqrt(delta))
print("As raízes são", raiz1, "e", raiz2)
```

As raízes são -1.0 e 4.0

Nomes (variáveis)

```
# Calcula raízes da equação de segundo grau
import math
a = -1
b = 3
c = 4
delta = b**2 - 4 * a * c  # Não pode ser menor que zero!
raiz1 = -2 * c / (b + math.sqrt(delta))
raiz2 = -2 * c / (b - math.sqrt(delta))
print("As raízes são", raiz1, "e", raiz2)
```

As raízes são -1.0 e 4.0

Nomes (variáveis)

- **Por que o título destes slides é “Nomes (variáveis)”?**

Nomes (variáveis)

- **Por que o título destes slides é “Nomes (variáveis)”?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos

Nomes (variáveis)

- **Por que o título destes slides é "Nomes (variáveis)"?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação "real" que está em algum lugar na memória do computador)
    - » *Como na matemática!*

Nomes (variáveis)

- **Por que o título destes slides é "Nomes (variáveis)"?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação "real" que está em algum lugar na memória do computador)
    - » *Como na matemática!*
  - Por isso, chamamos esses nomes de "variáveis"

Nomes (variáveis)

- **Por que o título destes slides é "Nomes (variáveis)"?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação "real" que está em algum lugar na memória do computador)
    - » *Como na matemática!*
  - Por isso, chamamos esses nomes de "variáveis"
  - Acima, definimos o valor dos nomes (variáveis) com comandos de atribuição (`a = -1`) fixos. Como fazer se queremos valores que não sejam fixos?

Nomes (variáveis)

- **Por que o título destes slides é "Nomes (variáveis)"?**
  - Não há muita graça em escrever programas como os que vimos acima, em que os dados são sempre os mesmos
  - Um dos principais usos de nomes é representar valores que *variam* (basicamente, alguma informação "real" que está em algum lugar na memória do computador)
    - » *Como na matemática!*
  - Por isso, chamamos esses nomes de "variáveis"
  - Acima, definimos o valor dos nomes (variáveis) com comandos de atribuição (`a = -1`) fixos. Como fazer se queremos valores que não sejam fixos?
  - `input()`

Primeiro programa – bully

```
idade = input("Informe sua idade: ")
```

Primeiro programa – bully

```
idade = input("Informe sua idade: ")

print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * idade, "anos!")
```

Primeiro programa – bully

```
idade = input("Informe sua idade: ")

print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * idade, "anos!")
```

Informe sua idade: 23

Primeiro programa – bully

```
idade = input("Informe sua idade: ")

print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * idade, "anos!")
```

Informe sua idade: 23

Você tem só 23 anos?!?!

Primeiro programa – bully

```
idade = input("Informe sua idade: ")

print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * idade, "anos!")
```

Informe sua idade: 23

Você tem só 23 anos?!?!

Nossa, você aparenta ter 2323 anos!

Primeiro programa – bully

```
idade = input("Informe sua idade: ")

print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * idade, "anos!")
```

Primeiro programa – bully

```
idade = int(input("Informe sua idade: "))

print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * idade, "anos!")
```

Primeiro programa – bully

```
idade = input("Informe sua idade: ")

print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * int(idade), "anos!")
```

Primeiro programa – bully

```
idade = input("Informe sua idade: ")
idade = int(idade)
print("Você tem só", idade, "anos?!?!")
print("Nossa, você aparenta ter", 2 * idade, "anos!")
```

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")
```

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")
```

- **`n % 2 == 0`** → **`True`** ou **`False`**

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")
```

- **`n % 2 == 0` → `True` ou `False`**
  - Embora possamos ler “se condição”, na verdade python faz “se o valor da expressão é verdadeiro (**`True`**)”

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")
```

- **`n % 2 == 0` → `True` ou `False`**
  - Embora possamos ler “se condição”, na verdade python faz “se o valor da expressão é verdadeiro (`True`)”
  - É **como se** python executasse `if condição == True`

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")
```

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")

print("Ahazei!")
```

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")

print("Ahazei!")
```

- **Ele sempre imprime “Ahazei!” ou só quando o número é par?**

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")

print("Ahazei!")
```

- Ele sempre imprime “Ahazei!” ou só quando o número é par?
- Como ele sabe onde “acaba o efeito” do `if`?

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")

print("Ahazei!")
```

- Ele sempre imprime “Ahazei!” ou só quando o número é par?
- Como ele sabe onde “acaba o efeito” do `if`?

Execução condicional

```
n = int(input("Digite um número natural: "))

if n % 2 == 0:
    print("O número", n, "é par!")

print("Ahazei!")
```

- **Ele sempre imprime “Ahazei!” ou só quando o número é par?**
- **Como ele sabe onde “acaba o efeito” do `if`?**
  - Qualquer quantidade de espaços, desde que seja consistente (4 espaços é o mais comum)

Execução condicional

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
if n % 2 != 0:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

Execução condicional

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
if n % 2 != 0:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **É muito comum que queiramos "cuidar" de todos os casos possíveis...**

Execução condicional

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
if n % 2 != 0:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **É muito comum que queiramos "cuidar" de todos os casos possíveis...**
  - Mas aqui só há dois casos possíveis!

Execução condicional

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
if n % 2 != 0:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **É muito comum que queiramos "cuidar" de todos os casos possíveis...**
  - Mas aqui só há dois casos possíveis!
    - » *(e eles são mutuamente excludentes)*

Execução condicional

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
if n % 2 != 0:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **É muito comum que queiramos "cuidar" de todos os casos possíveis...**
  - Mas aqui só há dois casos possíveis!
    - » *(e eles são mutuamente excludentes)*
  - Para que verificar a mesma condição duas vezes?

Execução condicional

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
if n % 2 != 0:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **É muito comum que queiramos "cuidar" de todos os casos possíveis...**
  - Mas aqui só há dois casos possíveis!
    - » *(e eles são mutuamente excludentes)*
  - Para que verificar a mesma condição duas vezes?
  - Isso não é muito fácil de ler!

Execução condicional – else

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
else:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

Execução condicional – `else`

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
else:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- A *indentação* indica os "lados" do condicional

Execução condicional – `else`

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
else:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **A *indentação* indica os "lados" do condicional**
  - Sem variável ("n"), o programa sempre executaria o mesmo "lado"

Execução condicional – `else`

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
else:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **A *indentação* indica os "lados" do condicional**
  - Sem variável ("n"), o programa sempre executaria o mesmo "lado"
- **O estado da variável só importa no momento do teste**

Execução condicional – else

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
else:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **A *indentação* indica os "lados" do condicional**
  - Sem variável ("n"), o programa sempre executaria o mesmo "lado"
- **O estado da variável só importa no momento do teste**
  - Se ela mudar em seguida, não afeta o condicional

Execução condicional – else

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
else:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **A *indentação* indica os "lados" do condicional**
  - Sem variável ("n"), o programa sempre executaria o mesmo "lado"
- **O estado da variável só importa no momento do teste**
  - Se ela mudar em seguida, não afeta o condicional
- **Os "lados" são mutuamente excludentes**

Execução condicional – `else`

```
n = int(input("Digite um número natural: "))
if n % 2 == 0:
    print("O número", n, "é par!")
else:
    print("O número", n, "é ímpar!")
print("Ahazei!")
```

- **A *indentação* indica os "lados" do condicional**
  - Sem variável ("n"), o programa sempre executaria o mesmo "lado"
- **O estado da variável só importa no momento do teste**
  - Se ela mudar em seguida, não afeta o condicional
- **Os "lados" são mutuamente excludentes**
  - **Um** e **apenas um** deles é executado

Repetições

Por que computação?

O computador é extremamente rápido, mas

- **É uma ferramenta com o mesmo nível de "inteligência" que um martelo**
  - Tudo tem que ser esmiuçado nos mínimos detalhes
    - » *"Vá à padaria e compre três pães"*
    - » *"Localize esta palavra no texto"*
    - » *...*

Não é mais fácil fazer manualmente?

Por que computação?

Às vezes, sim 🙂 mas:

- **Sistemas de controle**
- **Comunicação**
- **...**
- **Repetições**

Repetições “externas” e “internas”

- **Algumas repetições são externas ao programa**
  - Calculadora (o usuário faz inúmeros cálculos)
  - Jogo (cada partida é uma “repetição”)
  - ...
- **Mas, em geral, qualquer programa não-trivial vai realizar repetições internamente**
  - Xadrez (cada jogada é uma repetição)
  - Procurar uma palavra em um texto (várias comparações)
  - Apresentar uma foto na tela (cada pixel precisa ser “pintado” com a cor adequada)
  - ...

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Repetições e condições

- **Em ambos os casos, "algo" precisa acontecer para indicar que as repetições chegaram ao fim**

  (ok, às vezes queremos repetir indefinidamente, mas vamos ignorar isso por enquanto)

- **As repetições são controladas por algum tipo de *condição* baseada no estado de uma *variável***

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Tipos de repetição

- **Dois tipos fundamentais de repetição**
  1. Repetições até atingir um resultado
     - » *Encontrar o próximo primo*
     - » *Reiniciar o jogo até o usuário escolher "sair"*
     - » *...*
  2. Repetições sobre os elementos de um conjunto
     - » *Apresentar todos os pixels de uma foto na tela*
     - » *Trocar todas as letras de um texto para maiúsculas*
     - » *...*

Repetições até atingir um resultado

```
usuarioQuerJogar = True
while usuarioQuerJogar:
    # Joga uma partida...
    resposta = input("Você quer jogar novamente? ")
    if resposta != "S":
        usuarioQuerJogar = False
print("Cabô!")
```

Repetições até atingir um resultado

```
usuarioQuerJogar = True
while usuarioQuerJogar:
    # Joga uma partida...
    resposta = input("Você quer jogar novamente? ")
    if resposta != "S":
        usuarioQuerJogar = False
print("Cabô!")
```

- **Sem variável, o programa nunca pararia de repetir**
  - (aqui, a variável é "`usuarioQuerJogar`")
- **A *condição* precisa mudar ao menos na última iteração!**
- **A *condição* testada é "`usuarioQuerJogar` é verdadeiro"**

Repetições até atingir um resultado

```
usuarioQuerJogar = True
while usuarioQuerJogar:
    # Joga uma partida...
    resposta = input("Você quer jogar novamente? ")
    if resposta != "S":
        usuarioQuerJogar = False
print("Cabô!")
```

- **Sem variável, o programa nunca pararia de repetir**
  - (aqui, a variável é "`usuarioQuerJogar`")
- **A *condição* precisa mudar ao menos na última iteração!**
- **A *condição* testada é "`usuarioQuerJogar` é verdadeiro"**
  - » *(Onde está "* `== True` *"?)*