MAC 115 — Introdução à Ciência da Computação

Aula 2

Nelson Lago

IF noturno – 2023

Programando

Programar envolve

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - Cálculos?
   - Armazenamento e recuperação de dados?
   - Processamento de multimídia?
   - ...

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - Cálculos?
   - Armazenamento e recuperação de dados?
   - Processamento de multimídia?
   - ...
2. **Definir como esse problema pode ser solucionado (*algoritmo*)**
   - O algoritmo é *abstrato* (como a planta de um prédio ou uma receita de bolo)

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - Cálculos?
   - Armazenamento e recuperação de dados?
   - Processamento de multimídia?
   - ...
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

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - Cálculos?
   - Armazenamento e recuperação de dados?
   - Processamento de multimídia?
   - ...
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
     - » *Uma receita de bolo em alemão é útil? Para quem?*

Programando

Programar envolve

1. **Compreender um problema em termos computacionais**
   - Cálculos?
   - Armazenamento e recuperação de dados?
   - Processamento de multimídia?
   - ...
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
     - » *Uma receita de bolo em alemão é útil? Para quem?*
4. **Testar o programa**

Programando

Para ser útil, um programa geralmente

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**

Programando

Para ser útil, um programa geralmente

1. **Obtém dados**
2. **"Faz alguma coisa" com esses dados**
   - Gerando um resultado

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**

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

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!

O IDLE

- **Vamos começar usando o *shell* do python através do IDLE**
  - O shell é um ambiente interativo para a execução de comandos python
- **Em geral, *não* se usa o shell; ele serve apenas para experimentar alguns comandos básicos**
- **O primeiro comando que vamos ver é `print()`**

print()

```
print("Olá!")
```

print()

```
print("Olá!")
```

Olá

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: 2 + 3

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: 2 + 3
  - Exemplo: "Oi galera!"

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: 2 + 3
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: 2 + 3
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: 2 + 3 + 7

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: $2 + 3$
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 + 3 + 7$
  - Exemplo: $\frac{2+3+7}{6}$

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: 47
  - Exemplo: 2 + 3
  - Exemplo: "Oi galera!"
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: 2 + 3 + 7
  - Exemplo: $\frac{2+3+7}{6}$

Expressões aritméticas

$$2 + 3 + 7$$

Expressões aritméticas

$$2 + 3 + 7$$

```
print(2 + 3 + 7)
```

Expressões aritméticas

$$2 + 3 + 7$$

```
print(2 + 3 + 7)
```

```
12
```

Expressões aritméticas

$$\frac{2+3+7}{6}$$

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
print(2 + 3 + 7 / 6)
```

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
print(2 + 3 + 7 / 6)
```

```
6.166666666666667
```

Oops!

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
print((2 + 3 + 7) / 6)
```

Expressões aritméticas

$$\frac{2+3+7}{6}$$

```
print((2 + 3 + 7) / 6)
```

```
2.0
```

Precedência de operadores

operador	descrição	associatividade
`()`	parênteses	da esquerda para a direita
`**`	exponenciação	da direita para a esquerda
`+`, `-`	positivo e negativo unário	da direita para a esquerda
`*`, `/`, `//`, `%`	multiplicação, divisão, divisão inteira e resto	da esquerda para a direita
`+`, `-`	soma e subtração	da esquerda para a direita

Precedência dos operadores aritméticos em python (da maior para a menor)

Tipos

```
print((2 + 3 + 7) / 6)
print(4 - 2)
print(4 - 2.0)
print(2 * 1)
```

Tipos

```
print((2 + 3 + 7) / 6)
print(4 - 2)
print(4 - 2.0)
print(2 * 1)
```

2.0

Tipos

```
print((2 + 3 + 7) / 6)
print(4 - 2)
print(4 - 2.0)
print(2 * 1)
```

2.0

2

Tipos

```
print((2 + 3 + 7) / 6)
print(4 - 2)
print(4 - 2.0)
print(2 * 1)
```

2.0

2

2.0

Tipos

```
print((2 + 3 + 7) / 6)
print(4 - 2)
print(4 - 2.0)
print(2 * 1)
```

2.0

2

2.0

2

Tipos

```
print((2 + 3 + 7) / 6)
print(4 - 2)
print(4 - 2.0)
print(2 * 1)
```

2.0

2

2.0

2

Tipos

```
print(2 + 3)
print("2 + 3")
print("2" + "3")
print("2" + 3)
```

Tipos

```
print(2 + 3)
print("2 + 3")
print("2" + "3")
print("2" + 3)
```

```
5
```

Tipos

```
print(2 + 3)
print("2 + 3")
print("2" + "3")
print("2" + 3)
```

```
5
2 + 3
```

Tipos

```
print(2 + 3)
print("2 + 3")
print("2" + "3")
print("2" + 3)
```

```
5
2 + 3
23
```

Tipos

```
print(2 + 3)
print("2 + 3")
print("2" + "3")
print("2" + 3)
```

5

2 + 3

23

[Erro]

Tipos

```
print(2 + 3)
print("2 + 3")
print("2" + "3")
print("2" + 3)
```

5

2 + 3

23

[Erro]



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

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
    - » *Mas como ele sabe quando usar qual?*

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
    - » *Mas como ele sabe quando usar qual?*
- **Se solicitarmos ao computador que calcule $\frac{2}{3}$, quanto de memória vai ser necessário para armazenar o resultado?**

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
    - » *Mas como ele sabe quando usar qual?*
- **Se solicitarmos ao computador que calcule $\frac{2}{3}$, quanto de memória vai ser necessário para armazenar o resultado?**
- **E $\sqrt{2}$?**

Tipos

- Existem *tipos de dados* diferentes em python

Tipos

- **Existem *tipos de dados* diferentes em python**
  - Números inteiros, como 2 ou -437
    - » *Em python, tão grandes quanto necessário (até o limite da memória disponível); em outras linguagens, normalmente há um tamanho máximo*

Tipos

- **Existem *tipos de dados* diferentes em python**
  - Números inteiros, como 2 ou -437
    - » *Em python, tão grandes quanto necessário (até o limite da memória disponível); em outras linguagens, normalmente há um tamanho máximo*
  - Números (potencialmente) não-inteiros ("números de ponto flutuante"), como 2.0 ou 6.166666666666667

Tipos

- **Existem *tipos de dados* diferentes em python**
  - Números inteiros, como 2 ou -437
    - » *Em python, tão grandes quanto necessário (até o limite da memória disponível); em outras linguagens, normalmente há um tamanho máximo*
  - Números (potencialmente) não-inteiros ("números de ponto flutuante"), como 2.0 ou 6.166666666666667
    - » *Às vezes o resultado de cálculos pode ser aproximado*

Tipos

- **Existem *tipos de dados* diferentes em python**
  - Números inteiros, como 2 ou -437
    - » *Em python, tão grandes quanto necessário (até o limite da memória disponível); em outras linguagens, normalmente há um tamanho máximo*
  - Números (potencialmente) não-inteiros ("números de ponto flutuante"), como 2.0 ou 6.166666666666667
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - » `print(0.1 + 0.1 + 0.1)`

      `0.30000000000000004`

Tipos

- **Existem *tipos de dados* diferentes em python**
  - Números inteiros, como 2 ou -437
    - » *Em python, tão grandes quanto necessário (até o limite da memória disponível); em outras linguagens, normalmente há um tamanho máximo*
  - Números (potencialmente) não-inteiros ("números de ponto flutuante"), como 2.0 ou 6.166666666666667
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - » `print(0.1 + 0.1 + 0.1)`

      `0.30000000000000004`
  - Textos ("strings" ou "cadeias de caracteres"), como `"Oi galera!"`

Tipos

- **Existem *tipos de dados* diferentes em python**
  - Números inteiros, como 2 ou -437
    - » *Em python, tão grandes quanto necessário (até o limite da memória disponível); em outras linguagens, normalmente há um tamanho máximo*
  - Números (potencialmente) não-inteiros ("números de ponto flutuante"), como 2.0 ou 6.166666666666667
    - » *Às vezes o resultado de cálculos pode ser aproximado*
    - » `print(0.1 + 0.1 + 0.1)`

      `0.30000000000000004`
  - Textos ("strings" ou "cadeias de caracteres"), como `"Oi galera!"`
- **Python "sabe" quais operações podem ser realizadas com cada tipo**

Tipos

- **Então por que `4 - 2.0` é `2.0`?**

Tipos

- **Então por que `4 - 2.0` é `2.0`?**
  - Como um dos operandos é um número não-inteiro (ponto flutuante), python *converte* o outro para ponto flutuante também (imagine que o cálculo poderia ser `4 - 2.5`)

Tipos

- **Então por que `4 - 2.0` é `2.0`?**
  - Como um dos operandos é um número não-inteiro (ponto flutuante), python *converte* o outro para ponto flutuante também (imagine que o cálculo poderia ser `4 - 2.5`)
- **Então por que `(2 + 3 + 7) / 6` é `2.0` se todos os operandos são inteiros?**

Tipos

- **Então por que `4 - 2.0` é `2.0`?**
  - Como um dos operandos é um número não-inteiro (ponto flutuante), python *converte* o outro para ponto flutuante também (imagine que o cálculo poderia ser `4 - 2.5`)
- **Então por que `(2 + 3 + 7) / 6` é `2.0` se todos os operandos são inteiros?**
  - No ensino fundamental, aprendemos que $\frac{7}{4} = 1$ com resto 3, mas depois aprendemos que $\frac{7}{4} = 1,75$
    - » *Em python, `/` é a segunda operação; a primeira é `//`*
    - » *(informação bônus: `%` resulta no resto da divisão)*

Tipos

- **Então por que `4 - 2.0` é `2.0`?**
  - Como um dos operandos é um número não-inteiro (ponto flutuante), python *converte* o outro para ponto flutuante também (imagine que o cálculo poderia ser `4 - 2.5`)
- **Então por que `(2 + 3 + 7) / 6` é `2.0` se todos os operandos são inteiros?**
  - No ensino fundamental, aprendemos que $\frac{7}{4} = 1$ com resto 3, mas depois aprendemos que $\frac{7}{4} = 1,75$
    - » *Em python, `/` é a segunda operação; a primeira é `//`*
    - » *(informação bônus: `%` resulta no resto da divisão)*
  - Como a operação solicitada foi a divisão de ponto flutuante, python *converte* os dois operandos para ponto flutuante

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **AAAAAHHHH!!!!!!!**

Expressões em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **AAAAAHHHH!!!!!!!**
- **Qual o resultado dessa heresia?**

Operadores relacionais

```
print(2 > 3)
```

Operadores relacionais

```
print(2 > 3)
```

```
False
```

Operadores relacionais

```
print(2 > 3)
```

```
False
```

Ufa!

Operadores relacionais

operador	descrição
==	igualdade
!=	desigualdade
>	maior
>=	maior ou igual ($\geq$)
<	menor
<=	menor ou igual ($\leq$)

Operadores relacionais em python

Programando em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)

Programando em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$

Programando em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**

Programando em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 + 3 + 7 < 4$

Programando em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 + 3 + 7 < 4$
  - Exemplo: $2 + 2 == 2 * 2$

Programando em python

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 + 3 + 7 < 4$
  - Exemplo: $2 + 2 == 2 * 2$

Operadores relacionais

```
print(2 + 3 + 7 < 4, 2 + 2 == 2 * 2)
```

Operadores relacionais

```
print(2 + 3 + 7 < 4, 2 + 2 == 2 * 2)
```

```
False True
```

Expressões lógicas

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)

Expressões lógicas

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$

Expressões lógicas

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**

Expressões lógicas

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 > 3$ **and** $5 > 4$

Expressões lógicas

- **A maioria das coisas em python são *expressões***
  - (expressões são coisas que têm um *valor*)
  - Exemplo: $2 > 3$
- **Expressões podem ser combinadas ou utilizadas como partes de outras expressões**
  - Exemplo: $2 > 3$ **and** $5 > 4$

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
```

Precedência de operadores

operador	descrição
`not`	negação lógica (“inverte o sinal”)
`and`	E lógico (só é verdadeiro se ambos os operandos são verdadeiros)
`or`	OU lógico (só é verdadeiro se ao menos um dos operandos é verdadeiro)

Precedência dos operadores lógicos em python (da maior para a menor)

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***

Tipos booleanos

- O que exatamente são `True` e `False`?

Tipos booleanos

- O que exatamente são `True` e `False`?
- São um outro *tipo* de dado (como inteiros, *strings* e números de ponto flutuante)

Tipos booleanos

- **O que exatamente são `True` e `False`?**
- **São um outro *tipo* de dado (como inteiros, *strings* e números de ponto flutuante)**
- **Esse tipo se chama *booleano* (em homenagem a George Boole)**
  - Tipos booleanos só podem ter dois valores: `True` e `False`

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'>
```

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'>
```