

MAC 110 – Introdução à Ciência da Computação

Aula 1

Nelson Lago

BMAC – 2024



- **Informações sobre a disciplina no eDisciplinas**

- ▶ Datas de provas e critérios de avaliação
- ▶ Slides das aulas (a cada aula)
- ▶ Exercícios para praticar
- ▶ Entrega dos exercícios que valem nota
- ▶ Links para muitos materiais adicionais interessantes
- ▶ Fórum de discussões e informações sobre a monitoria
- ▶ O eDisciplinas pode enviar avisos por email; não deixe de checar sua caixa postal regularmente

Fazer muitos exercícios é fundamental



MAC110 – Introdução à Computação (BMAC 2024)

edisciplinas.usp.br/course/view.php?id=117555

Plágio/cola/etc. não dá, né?

**Leia o item sobre “plágio++”
no eDisciplinas**

Computação:

Resolução de problemas do mundo real com os recursos que o computador oferece

- **Desenvolver o raciocínio aplicado na formulação e resolução de problemas computacionais**
 - ▶ Aprender a escrever programas de computador (na linguagem Python)
- **Mas não é um “curso de Python”!**
 - ▶ O ensino de uma linguagem de programação é uma ferramenta didática e um bônus concreto de aprendizagem

- **Instale o python e o IDLE: python.org**
 - ▶ Há vídeos com dicas de como fazer no eDisciplinas, mas é bem simples
- **Se você quiser, pode usar outros ambientes ao invés do IDLE (Spyder, VSCode etc.)**

Resolução de problemas do mundo real com os recursos que o computador oferece

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)
 - » *E não tem vontade própria!* (“o sistema não permite...”)

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

Um exemplo de algoritmo

$$\begin{array}{r} 123 \\ \times 45 \\ \hline 615 \\ 492+ \\ \hline 5535 \end{array}$$

Programar também envolve

❶ Construir software capaz de lidar com dificuldades especiais

- ▶ Desempenho (sistemas de tempo real, HPC...)
- ▶ Grandes quantidades de dados (big data, aprendizado de máquina...)
- ▶ Tolerância a falhas (*checkpointing*, sistemas distribuídos...)
- ▶ Segurança (ataques cibernéticos, privacidade...)

❷ Gerenciar software de grande porte, composto por várias partes que precisam ser integradas

❸ Gerenciar o processo e a equipe de desenvolvimento

❹ Comunicar-se com clientes e usuários

Para ser útil, um programa geralmente

❶ **Obtém dados**

❷ **“Faz alguma coisa” com esses dados**

- ▶ Gerando um resultado

❸ **“Faz alguma coisa” com esse resultado**

- ▶ Mostra para o usuário

- ▶ **Utiliza como dado para fazer outra coisa**

Lembre-se:

O computador só faz o que você manda!

(não o que você quer)

Se você esquecer um passo, ele obedece!

- **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!")
```

Olá

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



$$2 + 3 + 7$$

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

12

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)

Brincando com dígitos

- **O operador `//` faz uma divisão inteira**
 - ▶ $13 // 5 = 2$
- **Se o divisor é 10, o resultado é que eliminamos o último dígito:**
 - ▶ $12345 // 10 = 1234$
- **O operador `%` fornece o resto da divisão**
 - ▶ $13 \% 5 = 3$
 - » *(Por definição, o resultado sempre é menor que o divisor)*
- **Se o divisor é 10, o resultado é igual ao último dígito:**
 - ▶ $12345 \% 10 = 5$
 - » *Qualquer número ...xyz pode ser escrito como ...xy * 10 + z*
 - » *...xy * 10 é múltiplo de 10, então o resto da divisão de ...xyz por 10 é z*
 - » *(e, como visto acima, z sempre é menor que 10)*

(Não é exatamente “divisão inteira” nem “resto da divisão”, mas ok)

Brincando com dígitos

Encontre o último dígito de um número:

```
print(2407 % 10)
print(110 % 10)
print(3 % 10)
print(0 % 10)
```

7

0

3

0

Brincando com dígitos

Dado um número n , transforme-o no formato $a + b$, onde $a + b = n$ e a é o maior múltiplo de 10 tal que $a \leq n$

$$b = n \% 10$$

$$a = n - b$$

Tipos

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

5

2 + 3

23

[Erro]

- **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 $\sqrt{2}$?**

- 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(math.sqrt(5) * math.sqrt(5))`

- 5.0000000000000001 *Ok, números irracionais...*

- `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

- **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 `//`
- ▶ 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*)
 - Exemplo: `2 > 3`
- AAAAAHHHH!!!!!!!
- Qual o resultado dessa heresia?

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

```
False
```

Ufa!

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"))
```

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

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*)
 - ▶ Exemplo: 47
 - ▶ Exemplo: 2 + 3
 - ▶ Exemplo: 2 > 3
- Expressões podem ser combinadas ou utilizadas como partes de outras expressões
 - ▶ Exemplo: $\frac{2+3+7}{6}$
 - ▶ Exemplo: 2 + 3 + 7 < 4
 - ▶ Exemplo: 2 + 2 == 2 * 2



Operadores relacionais

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

False True

Os operandos são inteiros, mas o resultado da expressão é um booleano

Expressões lógicas

- A maioria das coisas em python são *expressões*
 - ▶ (expressões são coisas que têm um *valor*)
 - ▶ Exemplo: 47
 - ▶ Exemplo: 2 + 3
 - ▶ Exemplo: 2 > 3
- Expressões podem ser combinadas ou utilizadas como partes de outras expressões
 - ▶ Exemplo: 2 + 3 + 7 < 4
 - ▶ 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)
```

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***

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 ← 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
```

expressão (valor)

nome

atribuição

AAAAAHHHH!!!!!!!

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'>

Expressões lógicas e tipos

Só vou à praia se:

- A temperatura no fim-de-semana for maior que 25°C
- Eu tiver pelo menos R\$100
- A minha nota na prova for A ou B

```
temp = 29
saldo = 100.39
nota = "B"
print(temp > 25 and saldo >= 100 and (nota == "A" or nota == "B"))
```

True

bool