MAC 110 — Introdução à Ciência da Computação

Aula 4

Nelson Lago

BMAC – 2024

Previously on MAC 110...

Programando

1. **algoritmo vs implementação**
2. **entrada de dados → processamento → resultado**
   - Mostra o resultado para o usuário
   - **Utiliza o resultado como dado para fazer outra coisa**
3. **Existem *tipos de dados* diferentes em python (*int*, *float*, *string*, *bool*...)**
4. **Expressões são coisas que têm um *valor* (de algum *tipo*)**
   - E podem ser combinadas ou utilizadas como partes de outras expressões

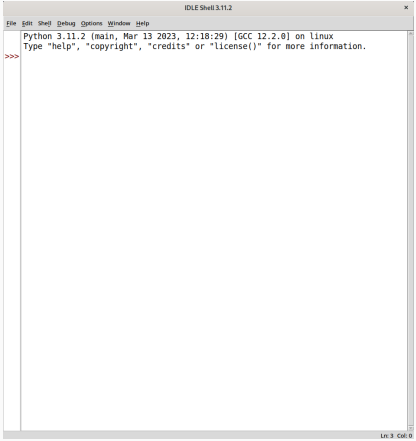
`2 + 3 + 7` (int)

`2 > 3 and 5 > 4` (bool)

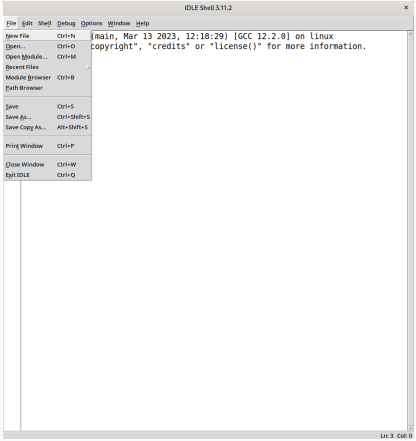
Adeus, shell!

- 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

VS


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

Nomes (variáveis)

- *Nomes* **permitem que pensemos mais sobre o problema a ser resolvido e menos sobre as idiossincrasias do computador**
- **Nomes em geral representam 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)

x ← 5 (atribuição)

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

```
x = 5
x = "Olá, galera!"
x = y
x = x + 1
```

Execução condicional 1/2

```
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 2/2

```
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 – `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
- **Com `else`, 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 (o objetivo foi atingido ou todos os elementos do conjunto já foram processados)**

  (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***

```
chute = input("Adivinhe qual é minha cor favorita: ")
while chute != "rosa choque":
    chute = input("Errou! Tente novamente: ")
print("Acertou!")
```

Partes mínimas de um laço

- **Um laço correto precisa**
  - Inicializar a variável de controle antes do início do laço
  - Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
  - Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

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

```
chute = input("Adivinhe qual é minha cor favorita: ")
while chute != "rosa choque":
    chute = input("Errou! Tente novamente: ")
print("Acertou!")
```

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

Partes mínimas de um laço

- **Um laço correto precisa**
  - Inicializar a variável de controle antes do início do laço
  - Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
  - Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

```
chute = input("Adivinhe qual é minha cor favorita: ")
while chute != "rosa choque":
    chute = input("Errou! Tente novamente: ")
print("Acertou!")
```

Repetições até atingir um resultado

- **Caso comum:**
  - branco sujo
  - amarelo icterícia
  - roxo hematoma
  - **rosa choque**

*A **variável** muda em todas as iterações, mas a **condição** só muda na última iteração*

- **Caso sortudo:**
  - **rosa choque**

***Nenhuma** iteração, então nem a variável nem a condição mudam*

- **Caso absurdo:**
  - roxo hematoma
  - roxo hematoma
  - roxo hematoma
  - **rosa choque**

*A variável **e** a condição só mudam na última iteração*

A variável sempre precisa mudar (ao menos) na última iteração!

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 sobre os elementos de um conjunto

```
print("Prepare-se para o grito:")
n = 10
while n >= 0:
    print(n)
    n = n -1
print("AAAHHH!!!!")
```

- **O *conjunto* são os números 0–10 (às vezes a ordem importa!)**
- **A *condição* precisa mudar ao menos na última iteração!**
  - (indicando que todos os elementos do conjunto foram processados)
    - » *A condição testada é "*`n` *é maior ou igual a zero"*
- **Sem variável, o programa nunca pararia de repetir**
  - (aqui, a variável é "n")
    - » *(Onde está "* `== True` *"?)*

Partes mínimas de um laço

- **Um laço correto precisa**
  - Inicializar a variável de controle antes do início do laço
  - Verificar a condição adequada a cada iteração para que as repetições aconteçam o número correto de vezes
  - Alterar o valor da variável de acordo com a lógica do programa (no mínimo, na última iteração) para garantir que o laço termine

```
print("Prepare-se para o grito:")
n = 10
while n >= 0:
    print(n)
    n = n -1
print("AAAHHH!!!!")
```

Exercícios

Exercício — dígito das dezenas

Dado um número, imprima o dígito das dezenas

```
n = int(input("Digite um número natural: "))
print("O dígito das dezenas é", (n % 100) // 10)
```

Exercício — ano bissexto

- **O que é um ano bissexto?**
  - O tempo de translação da terra ao redor do sol não é exatamente 365 dias; assim, a cada 4 anos, temos um ano bissexto para compensar essa diferença
  - No entanto, essa compensação não é perfeita; por conta disso, a cada 100 anos, um ano que normalmente seria bissexto não é
  - Essa segunda compensação também não é perfeita e, por isso, a cada 400 anos, um ano que excepcionalmente deixaria de ser bissexto é bissexto normalmente
    - » *Um ano é bissexto se é múltiplo de 4, exceto quando é múltiplo de 100 mas não de 400*
    - » *Um ano é bissexto se é múltiplo de 400 ou se é múltiplo de 4 mas não de 100*
    - » *Um ano é bissexto se é múltiplo de 4 mas não de 100, exceto se for múltiplo de 400*

Exercício — ano bissexto

Dado um ano, informar se ele é bissexto ou não

```
ano = int(input("Digite o ano: "))
if (ano % 4 == 0 and ano % 100 != 0) or ano % 400 == 0:
    print("O ano é bissexto")
else:
    print("O ano não é bissexto")
```

Exercício — estações do ano

Dada uma data (dia e mês), informe se a data acontece no verão (21 de dezembro a 20 de março), outono (21 de março a 20 de junho), inverno (21 de junho a 22 de setembro) ou primavera (23 de setembro a 20 de dezembro). Use números de 1 a 12 para representar os meses do ano.

```
dia = int(input("Digite o dia: "))
mês = int(input("Digite o mês (1 a 12): "))
dia_no_ano = (mês-1) * 30 + dia
if dia_no_ano <= 80 or dia_no_ano >= 351:
    print("Verão!")
if dia_no_ano > 80 and dia_no_ano <= 170:
    print("Outono!")
if dia_no_ano > 170 and dia_no_ano <= 262:
    print("Inverno!")
if dia_no_ano > 262 and dia_no_ano <= 350:
    print("Primavera!")
```

Truque sujo!
(funciona, mas...)

Exercício — contagem de pares

Leia uma série de números terminada por zero fornecida pelo usuário e diga quantos deles são pares

```
pares = 0
acabou = False
while not acabou:
    n = int(input("Digite um número (zero para sair): "))
    if n == 0:
        acabou = True
    else:
        if n % 2 == 0:
            pares = pares +1
print("Você digitou", pares, "números pares")
```

Exercício — contagem de pares e ímpares

Leia uma série de números terminada por zero fornecida pelo usuário e diga quantos deles são pares e quantos são ímpares

```
pares = 0
todos = 0
n = int(input("Digite um número (zero para sair): "))
while n != 0:
    if n % 2 == 0:
        pares = pares +1

    todos = todos +1
    n = int(input("Digite um número (zero para sair): "))
print("Você digitou", pares, "números pares e", todos - pares, "números ímpares")
```

O módulo turtle

Python inclui o módulo **turtle**, que permite desenhar figuras simples. Por exemplo, podemos desenhar um quadrado assim:

```
import turtle
michelangelo = turtle.Turtle()
michelangelo.forward(100)
michelangelo.right(90)
michelangelo.forward(100)
michelangelo.right(90)
michelangelo.forward(100)
michelangelo.right(90)
michelangelo.forward(100)
turtle.done()
```

Exercício — desenhando um quadrado

Modifique o programa anterior para desenhar um quadrado usando um laço

```
import turtle
michelangelo = turtle.Turtle()

lado = 1
while lado <= 4:
    michelangelo.forward(100)
    michelangelo.right(90)
    lado = lado +1
turtle.done()
```

Exercício — desenhando um círculo

Usando os comandos vistos do módulo **turtle**, desenhe um círculo

```
import turtle
arquimedes = turtle.Turtle()
arquimedes.width(4)

conta = 1
while conta <= 360:
    arquimedes.forward(1.5)
    arquimedes.right(1)
    conta = conta +1
turtle.done()
```

Exercício — desenhando uma estrela 1/3

Usando os comandos vistos do módulo **turtle**, desenhe uma estrela de cinco pontas: ☆

- $\frac{360^\circ}{5} = 72^\circ$

```
import turtle
ninja = turtle.Turtle()
ninja.width(4)

ponta = 1
while ponta <= 5:
    ninja.forward(100)
    ninja.right(72)
    ponta = ponta +1
turtle.done()
```

Exercício — desenhando uma estrela 2/3

Exercício — desenhando uma estrela 3/3

Usando os comandos vistos do módulo **turtle**, desenhe uma estrela de cinco pontas: 

- $\frac{2 \cdot 360^\circ}{5} = \frac{720^\circ}{5} = 144^\circ$

```
import turtle
ninja = turtle.Turtle()
ninja.width(4)

ponta = 1
while ponta <= 5:
    ninja.forward(100)
    ninja.right(144)
    ponta = ponta +1
turtle.done()
```