Estruturas condicionais e variáveis

Aula 2

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação

Conteúdo da aula

1. Motivação
2. Estrutura condicional simples
3. Estrutura condicional composta
4. Estrutura condicional composta (2)
5. Troca de conteúdos
6. Revisão de variáveis
7. Exercícios

Motivação

- instruções são sempre executadas uma após a outra?
- há possibilidade de seleção de um grupo de instruções de acordo com alguma situação que ocorre durante a execução de um programa?
- estruturas condicionais permitem tomadas de decisão, com desvio de fluxo de execução;
- estruturas condicionais fornecem maior poder de programação

Motivação

- instruções são sempre executadas uma após a outra?
- há possibilidade de seleção de um grupo de instruções de acordo com alguma situação que ocorre durante a execução de um programa?
- estruturas condicionais permitem tomadas de decisão, com desvio de fluxo de execução;
- estruturas condicionais fornecem maior poder de programação

Motivação

- instruções são sempre executadas uma após a outra?
- há possibilidade de seleção de um grupo de instruções de acordo com alguma situação que ocorre durante a execução de um programa?
- estruturas condicionais permitem tomadas de decisão, com desvio de fluxo de execução;
- estruturas condicionais fornecem maior poder de programação

Motivação

- instruções são sempre executadas uma após a outra?
- há possibilidade de seleção de um grupo de instruções de acordo com alguma situação que ocorre durante a execução de um programa?
- estruturas condicionais permitem tomadas de decisão, com desvio de fluxo de execução;
- estruturas condicionais fornecem maior poder de programação

Estrutura condicional simples

Formato geral:

```
if condição:
    ⋮
    bloco de instruções "dentro" do if
    ⋮
bloco de instruções "fora" do if
```

A **indentação** pode ser definida com qualquer quantidade de espaços, desde que a quantidade seja a mesma. Em geral utilizamos 2 ou 4 espaços.
A indentação é **necessária** para definir o trecho de código que compõe o `if`.

Estrutura condicional simples

- uma decisão é tomada de acordo com a avaliação da *condição*, que é uma expressão lógica;
- caso o resultado dessa avaliação seja *verdadeiro*, a instrução, ou o bloco de instruções, será executada(o); caso contrário, isto é, se o resultado da avaliação é *falso*, será ignorada(o).

Estrutura condicional simples

- uma decisão é tomada de acordo com a avaliação da *condição*, que é uma expressão lógica;
- caso o resultado dessa avaliação seja *verdadeiro*, a instrução, ou o bloco de instruções, será executada(o); caso contrário, isto é, se o resultado da avaliação é *falso*, será ignorada(o).

Estrutura condicional simples

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Recebe um número inteiro positivo que representa uma idade
# e emite uma mensagem na saída se a idade é inferior a 30

idade = int(input("Quantos anos você tem? "))

if idade < 30:
   print("Puxa! Você é bem jovem!")

print("Até breve!")

exit(0)
```

Estrutura condicional composta

Formato geral:

```
if condição:
    ⋮
    primeiro bloco de instruções
    ⋮
else:
    ⋮
    segundo bloco de instruções
    ⋮
```

Como nas estruturas condicionais simples, é preciso observar a indentação!

Estrutura condicional composta

- uma decisão é tomada de acordo com a avaliação da *condição*, que é uma expressão lógica;
- caso o resultado dessa avaliação seja *verdadeiro*, o primeiro bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada;
- caso contrário, isto é, se o resultado da avaliação da expressão lógica é *falso*, o segundo bloco de instruções, logo após a palavra-chave `else`, será executado; e ao final da execução das instruções desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada.

Estrutura condicional composta

- uma decisão é tomada de acordo com a avaliação da *condição*, que é uma expressão lógica;
- caso o resultado dessa avaliação seja *verdadeiro*, o primeiro bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada;
- caso contrário, isto é, se o resultado da avaliação da expressão lógica é *falso*, o segundo bloco de instruções, logo após a palavra-chave `else`, será executado; e ao final da execução das instruções desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada.

Estrutura condicional composta

- uma decisão é tomada de acordo com a avaliação da *condição*, que é uma expressão lógica;
- caso o resultado dessa avaliação seja *verdadeiro*, o primeiro bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada;
- caso contrário, isto é, se o resultado da avaliação da expressão lógica é *falso*, o segundo bloco de instruções, logo após a palavra-chave `else`, será executado; e ao final da execução das instruções desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada.

Estrutura condicional composta

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Recebe um número inteiro e emite uma
# mensagem de acordo com esse número

idade = int(input("Quantos anos você tem? "))

if idade < 30:
   print("Puxa! Você é bem jovem!")
else:
   print("Você já não é tão jovem!")
print("Até breve!")

exit(0)
```

Estrutura condicional composta (2)

Formato geral:

```
if condição:
    ⋮
    primeiro bloco de instruções
    ⋮
elif condição:
    ⋮
    segundo bloco de instruções
    ⋮
⋮
else:
    ⋮
    último bloco de instruções
    ⋮
```

Estrutura condicional composta (2)

- caso o resultado dessa avaliação seja *verdadeiro*, o primeiro bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada;
- caso contrário (**apenas se o resultado da avaliação do primeiro bloco for falso!**), o resultado da avaliação do segundo bloco de instruções é verificado, e caso o resultado dessa avaliação seja *verdadeiro*, o segundo bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada; ...
- caso o resultado da avaliação de todos os blocos for *falso*, o último bloco de instruções, logo após a palavra-chave `else`, será executado; e ao final da execução das instruções desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada.

Estrutura condicional composta (2)

- caso o resultado dessa avaliação seja *verdadeiro*, o primeiro bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada;
- caso contrário (**apenas se o resultado da avaliação do primeiro bloco for falso!**), o resultado da avaliação do segundo bloco de instruções é verificado, e caso o resultado dessa avaliação seja *verdadeiro*, o segundo bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada; ...
- caso o resultado da avaliação de todos os blocos for *falso*, o último bloco de instruções, logo após a palavra-chave `else`, será executado; e ao final da execução das instruções desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada.

Estrutura condicional composta (2)

- caso o resultado dessa avaliação seja *verdadeiro*, o primeiro bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada;
- caso contrário (**apenas se o resultado da avaliação do primeiro bloco for falso!**), o resultado da avaliação do segundo bloco de instruções é verificado, e caso o resultado dessa avaliação seja *verdadeiro*, o segundo bloco de instruções será executado e, ao término desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada; ...
- caso o resultado da avaliação de todos os blocos for *falso*, o último bloco de instruções, logo após a palavra-chave `else`, será executado; e ao final da execução das instruções desse bloco, a instrução da próxima linha após a estrutura condicional composta será executada.

Estrutura condicional composta (2)

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Recebe um número inteiro e emite uma
# mensagem de acordo com esse número

idade = int(input("Quantos anos você tem? "))

if idade < 30:
   print("Puxa! Você é bem jovem!")
elif idade > 60:
   print("Você já é um pouco mais velho!")
else:
   print("Você não é jovem nem velho!")
print("Até breve!")

exit(0)
```

Troca de conteúdos

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Recebe dois números inteiros x e y e coloca o menor desses
# valores em x e o maior em y, mostrando o resultado na saída

x = int(input("Informe o valor de x: "))
y = int(input("Informe o valor de y: "))
if x > y:
   x, y = y, x

printf("%d é menor ou igual a %d" % (x, y))

exit(0)
```

Revisão de variáveis

- durante a execução de programas podemos usar compartimentos de memória para armazenamento de informações;
- **variáveis**: valores podem ser atribuídos e sobrescritos nesses compartimentos e, portanto, podem variar durante a execução do programa (inclusive os tipos dos valores armazenados);
- regra fundamental sobre variáveis em programas: *qualquer variável é criada ao armazenar um valor pela primeira vez*, tente iniciar um programa com a instrução `var1 = var2`;
- podemos atribuir um nome simbólico a um endereço de um compartimento de memória: **identificador** ou **nome** da variável;
- o identificador de uma variável pode ser escolhido pelo(a) programador(a) de modo a refletir de alguma forma o seu conteúdo.

Revisão de variáveis

- durante a execução de programas podemos usar compartimentos de memória para armazenamento de informações;
- **variáveis**: valores podem ser atribuídos e sobrescritos nesses compartimentos e, portanto, podem variar durante a execução do programa (inclusive os tipos dos valores armazenados);
- regra fundamental sobre variáveis em programas: *qualquer variável é criada ao armazenar um valor pela primeira vez*, tente iniciar um programa com a instrução `var1 = var2`;
- podemos atribuir um nome simbólico a um endereço de um compartimento de memória: **identificador** ou **nome** da variável;
- o identificador de uma variável pode ser escolhido pelo(a) programador(a) de modo a refletir de alguma forma o seu conteúdo.

Revisão de variáveis

- durante a execução de programas podemos usar compartimentos de memória para armazenamento de informações;
- **variáveis**: valores podem ser atribuídos e sobrescritos nesses compartimentos e, portanto, podem variar durante a execução do programa (inclusive os tipos dos valores armazenados);
- regra fundamental sobre variáveis em programas: *qualquer variável é criada ao armazenar um valor pela primeira vez*, tente iniciar um programa com a instrução `var1 = var2`;
- podemos atribuir um nome simbólico a um endereço de um compartimento de memória: **identificador** ou **nome** da variável;
- o identificador de uma variável pode ser escolhido pelo(a) programador(a) de modo a refletir de alguma forma o seu conteúdo.

Revisão de variáveis

- durante a execução de programas podemos usar compartimentos de memória para armazenamento de informações;
- **variáveis**: valores podem ser atribuídos e sobrescritos nesses compartimentos e, portanto, podem variar durante a execução do programa (inclusive os tipos dos valores armazenados);
- regra fundamental sobre variáveis em programas: *qualquer variável é criada ao armazenar um valor pela primeira vez*, tente iniciar um programa com a instrução `var1 = var2`;
- podemos atribuir um nome simbólico a um endereço de um compartimento de memória: **identificador** ou **nome** da variável;
- o identificador de uma variável pode ser escolhido pelo(a) programador(a) de modo a refletir de alguma forma o seu conteúdo.

Revisão de variáveis

- durante a execução de programas podemos usar compartimentos de memória para armazenamento de informações;
- **variáveis**: valores podem ser atribuídos e sobrescritos nesses compartimentos e, portanto, podem variar durante a execução do programa (inclusive os tipos dos valores armazenados);
- regra fundamental sobre variáveis em programas: *qualquer variável é criada ao armazenar um valor pela primeira vez*, tente iniciar um programa com a instrução `var1 = var2`;
- podemos atribuir um nome simbólico a um endereço de um compartimento de memória: **identificador** ou **nome** da variável;
- o identificador de uma variável pode ser escolhido pelo(a) programador(a) de modo a refletir de alguma forma o seu conteúdo.

Revisão de variáveis

- pequeno conjunto de regras para um(a) programador(a) determinar os identificadores das suas variáveis:
  - os símbolos válidos em um identificador de uma variável são os caracteres alfabéticos, minúsculos ou maiúsculos, os dígitos numéricos e o sublinhado ou (_);
  - o identificador de uma variável deve iniciar com uma letra do alfabeto ou com um sublinhado (_);
  - após o primeiro caractere, o identificador de uma variável é determinado por qualquer sequência de letras minúsculas ou maiúsculas, de números ou de sublinhados.

Revisão de variáveis

- pequeno conjunto de regras para um(a) programador(a) determinar os identificadores das suas variáveis:
  - os símbolos válidos em um identificador de uma variável são os caracteres alfabéticos, minúsculos ou maiúsculos, os dígitos numéricos e o sublinhado ou (_);
  - o identificador de uma variável deve iniciar com uma letra do alfabeto ou com um sublinhado (_);
  - após o primeiro caractere, o identificador de uma variável é determinado por qualquer sequência de letras minúsculas ou maiúsculas, de números ou de sublinhados.

Revisão de variáveis

- pequeno conjunto de regras para um(a) programador(a) determinar os identificadores das suas variáveis:
  - os símbolos válidos em um identificador de uma variável são os caracteres alfabéticos, minúsculos ou maiúsculos, os dígitos numéricos e o sublinhado ou (_);
  - o identificador de uma variável deve iniciar com uma letra do alfabeto ou com um sublinhado (_);
  - após o primeiro caractere, o identificador de uma variável é determinado por qualquer sequência de letras minúsculas ou maiúsculas, de números ou de sublinhados.

Revisão de variáveis

identificadores de variáveis válidos:

```
soma
num1
i
soma_total
_sistema
A3x3
fracao
```

Revisão de variáveis

identificadores de variáveis **não-válidos**:

```
preço$
soma total
4quant
int
```

Revisão de variáveis

- letras minúsculas e maiúsculas na linguagem Python são diferentes: as variáveis `soma`, `Soma` e `SOMA` são todas diferentes;

Revisão de variáveis

- Podemos atribuir um mesmo valor a várias variáveis:

```
i = num1 = num2 = soma = produto = 0
```

- E também podemos atribuir diferentes valores para diferentes variáveis de uma só vez:

```
i, num1, num2, soma, produto = 0, -2, 5, 0, 1
```

- ou ainda, uma por vez:

```
i = 0
num1 = -2
num2 = 5
soma = 0
produto = 1
```

Revisão de variáveis

- Podemos atribuir um mesmo valor a várias variáveis:

```
i = num1 = num2 = soma = produto = 0
```

- E também podemos atribuir diferentes valores para diferentes variáveis de uma só vez:

```
i, num1, num2, soma, produto = 0, -2, 5, 0, 1
```

- ou ainda, uma por vez:

```
i = 0
num1 = -2
num2 = 5
soma = 0
produto = 1
```

Revisão de variáveis

- Podemos atribuir um mesmo valor a várias variáveis:

```
i = num1 = num2 = soma = produto = 0
```

- E também podemos atribuir diferentes valores para diferentes variáveis de uma só vez:

```
i, num1, num2, soma, produto = 0, -2, 5, 0, 1
```

- ou ainda, uma por vez:

```
i = 0
num1 = -2
num2 = 5
soma = 0
produto = 1
```

Revisão de variáveis

- identificadores de variáveis podem ser tão longos quanto se queira;
- porém, um programa escrito com identificadores de variáveis muito longos pode ser difícil de ser escrito e compreendido:

```
TotalDeDinheiroQueTenho = TotalDoDinheiroQueGuardeiAnoPassado +
      TotalDeDinheiroQueGuardeiEsteAno - TotalDeImpostosEmReais;
```

```
TotalGeral = TotalAno + TotalAnterior - Impostos;
```

Revisão de variáveis

- identificadores de variáveis podem ser tão longos quanto se queira;
- porém, um programa escrito com identificadores de variáveis muito longos pode ser difícil de ser escrito e compreendido:

```
TotalDeDinheiroQueTenho = TotalDoDinheiroQueGuardeiAnoPassado +
      TotalDeDinheiroQueGuardeiEsteAno - TotalDeImpostosEmReais;
```

```
TotalGeral = TotalAno + TotalAnterior - Impostos;
```

Revisão de variáveis

- identificadores de variáveis podem ser tão longos quanto se queira;
- porém, um programa escrito com identificadores de variáveis muito longos pode ser difícil de ser escrito e compreendido:

```
TotalDeDinheiroQueTenho = TotalDoDinheiroQueGuardeiAnoPassado +
      TotalDeDinheiroQueGuardeiEsteAno - TotalDeImpostosEmReais;
```

```
TotalGeral = TotalAno + TotalAnterior - Impostos;
```

Revisão de variáveis com valores inteiros

- linguagem Python possui diversos operadores aritméticos binários que podemos usar com números inteiros:
  - `+` para adição;
  - `-` para subtração;
  - `*` para multiplicação;
  - `/` para quociente da divisão;
  - `//` para quociente inteiro da divisão;
  - `%` para resto da divisão; e
  - `**` para potência (incluindo expoentes reais).
- adicionalmente, temos a função `abs(`*valor*`)`
- temos também o operador unário `-`, que devolve o valor de uma variável com o sinal invertido

Revisão de variáveis com valores inteiros

- linguagem Python possui diversos operadores aritméticos binários que podemos usar com números inteiros:
  - `+` para adição;
  - `-` para subtração;
  - `*` para multiplicação;
  - `/` para quociente da divisão;
  - `//` para quociente inteiro da divisão;
  - `%` para resto da divisão; e
  - `**` para potência (incluindo expoentes reais).
- adicionalmente, temos a função **`abs`**`(`*valor*`)`
- temos também o operador unário `-`, que devolve o valor de uma variável com o sinal invertido

Revisão de variáveis

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Realiza operações aritméticas com números */

y = 10
r = 25 + y
print("A soma de %d e %d é %d" % (25, y, r))
x = 38
r = x - 25
print("A subtração de %d e %d é %d" % (x, 25, r))
x = 51
y = 17
r = x * y
print("A multiplicação de %d por %d é %d" % (x, y, r))
x = 90
y = 4
r = x / y
print("O quociente da divisão de %d por %d é %d" % (x, y, r))
x = 90
y = 4
r = x // y
print("O quociente inteiro da divisão de %d por %d é %d" % (x, y, r))
```

Revisão de variáveis

```
x = 17
y = 3
r = x % y
print("O resto da divisão de %d por %d é %d" % (x, y, r))
x = -7
r = -x
print("%d com sinal trocado é %d" % (x, r))
r = abs(x)
print("O valor absoluto de %d é %d" % (x, r))
x = 2
y = 3
r = x ** y
print("%d à %d potência é %d" % (x, y, r))
x = 10
y = 4
z = 15
r = x + y * z
print("A expressão %d+%d*%d é %d" % (x, y, z, r))

exit(0)
```

Revisão de variáveis

- o resultado da avaliação da expressão aritmética `10 + 4 * 15` não é $14 \times 15 = 210$, mas sim $10 + 60 = 70$;
- operadores aritméticos na linguagem Python têm precedências uns sobre os outros;
- o resultado da expressão:

```
x + y * z
```

é dado primeiro pela avaliação da multiplicação e depois pela avaliação da adição;

Revisão de variáveis

- o resultado da avaliação da expressão aritmética `10 + 4 * 15` não é $14 \times 15 = 210$, mas sim $10 + 60 = 70$;
- operadores aritméticos na linguagem Python têm precedências uns sobre os outros;
- o resultado da expressão:

```
x + y * z
```

é dado primeiro pela avaliação da multiplicação e depois pela avaliação da adição;

Revisão de variáveis

- o resultado da avaliação da expressão aritmética `10 + 4 * 15` não é $14 \times 15 = 210$, mas sim $10 + 60 = 70$;
- operadores aritméticos na linguagem Python têm precedências uns sobre os outros;
- o resultado da expressão:

```
x + y * z
```

é dado primeiro pela avaliação da multiplicação e depois pela avaliação da adição;

Revisão de variáveis

Operador	Descrição	Precedência
`**`	Exponenciação	1 (máxima)
`* / %`	Multiplicação, quociente da divisão e resto da divisão	2
`+ -`	Adição e subtração	3 (mínima)

Exercícios

1. Faça a simulação passo a passo dos programas desta aula.

Exercícios

1. Faça a simulação passo a passo dos programas desta aula.

Exercícios

2. O programa a seguir recebe a temperatura ambiente em um dado instante em graus Célsius e mostra uma mensagem para o(a) usuário(a) informando se a temperatura está muito quente, considerando como temperatura limite o valor de 30 graus Célsius. Faça a simulação passo a passo da execução da solução deste programa.

Exercícios

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Recebe um número inteiro que representa uma
# temperatura e imprime uma mensagem de aviso

temperatura = int(input("Informe uma temperatura (em graus Célsius): "))

if temperatura > 30:
   print("Hoje é um dia bem quente!")
else:
   print("Hoje não é um dia tão quente.")

exit(0)
```

Exercícios

3. Polinômios são usados em uma ampla variedade de áreas da Matemática e das Ciências. Um **polinômio** é uma expressão de comprimento finito construída a partir de variáveis e constantes, usando apenas as operações de adição, subtração, multiplicação e expoentes não-negativos. Por exemplo, $x^3 + x^2 - 8$ é um polinômio.
   Uma **função polinomial** é uma função que pode ser definida pela avaliação de um polinômio. Uma função f, de um único argumento, é chamada uma função polinomial se satisfaz:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0$$

   para qualquer x, onde n é um número inteiro não-negativo e $a_0, a_1, \ldots, a_n$ são coeficientes constantes.

Exercícios

3. (*continuação*)
   Por exemplo, a função $f : \mathbb{Z} \to \mathbb{Z}$ definida por

$$f(x) = x^3 + x^2 - 8$$

   é uma função polinomial de um argumento (x).
   Escreva um programa que receba um número inteiro x e avalie a função polinomial

$$p(x) = 3x^3 - 5x^2 + 2x - 1\ .$$

   Faça a simulação passo a passo da execução de sua solução.

Exercícios

4. Para transformar um número inteiro i no menor inteiro m maior que i e múltiplo de um número inteiro j, a seguinte fórmula pode ser utilizada:

$$m = i + j - i \bmod j \,,$$

onde o operador mod é o operador de resto de divisão inteira na notação matemática usual, que corresponde ao nosso operador `%`.

Por exemplo, suponha que usamos $i = 256$ dias para alguma atividade e queremos saber qual o total de dias m que devemos ter de forma que esse número seja divisível por $j = 7$, para termos uma ideia do número de semanas que usaremos na atividade. Então, pela fórmula acima, temos que

$$\begin{aligned} m &= 256 + 7 - 256 \bmod 7 \\ &= 256 + 7 - 4 \\ &= 259 \,. \end{aligned}$$

Escreva um programa que receba dois números inteiros positivos i e j e devolva o menor inteiro m maior que i e múltiplo de j. Faça a simulação passo a passo da execução de sua solução.

Exercícios

5. Escreva um programa que receba um número inteiro a e verifique se a é par ou ímpar. Faça a simulação passo a passo da execução de sua solução.
6. Escreva um programa que receba um número inteiro a e verifique se a é positivo, se a é negativo ou se a é igual a 0. Faça a simulação passo a passo da execução de sua solução.
7. Escreva um programa que receba três valores, armazenando-os nas variáveis x, y e z, e ordene esses valores de modo que, ao final, o menor valor esteja armazenado na variável x, o valor intermediário esteja armazenado na variável y e o maior valor esteja armazenado na variável z. Faça a simulação passo a passo da execução de sua solução.

Exercícios

5. Escreva um programa que receba um número inteiro a e verifique se a é par ou ímpar. Faça a simulação passo a passo da execução de sua solução.
6. Escreva um programa que receba um número inteiro a e verifique se a é positivo, se a é negativo ou se a é igual a 0. Faça a simulação passo a passo da execução de sua solução.
7. Escreva um programa que receba três valores, armazenando-os nas variáveis x, y e z, e ordene esses valores de modo que, ao final, o menor valor esteja armazenado na variável x, o valor intermediário esteja armazenado na variável y e o maior valor esteja armazenado na variável z. Faça a simulação passo a passo da execução de sua solução.

Exercícios

5. Escreva um programa que receba um número inteiro a e verifique se a é par ou ímpar. Faça a simulação passo a passo da execução de sua solução.
6. Escreva um programa que receba um número inteiro a e verifique se a é positivo, se a é negativo ou se a é igual a 0. Faça a simulação passo a passo da execução de sua solução.
7. Escreva um programa que receba três valores, armazenando-os nas variáveis x, y e z, e ordene esses valores de modo que, ao final, o menor valor esteja armazenado na variável x, o valor intermediário esteja armazenado na variável y e o maior valor esteja armazenado na variável z. Faça a simulação passo a passo da execução de sua solução.

Exercícios

8. Escreva um programa que receba dois tempos no formato `hh:mm:ss` (um tempo por linha), some os dois tempos e escreva o tempo resultante. Por exemplo, para os tempos 03:10:32 e 04:55:40, você deve escrever na tela 08:06:12. Dica: para ler os tempos, você vai precisar utilizar `input.split()` de uma forma diferente da que usamos até agora, procure informações sobre a função `split()` do Python.