Capítulo 10

Comandos de seleção

Antes de apresentarmos mais alguns comandos, precisamos saber que existe um tipo de dado em Python chamado `bool` (de booleano). Diferente de números que podem assumir uma infinidade de valores, os objetos booleanos possuem apenas dois valores possíveis: verdadeiro (`True`) ou falso (`False`).

Esses valores são utilizados quando precisamos verificar se uma determina condição é satisfeita ou não. Por exemplo, para verificarmos se uma equação é do segundo grau, precisamos verificar se a variável `a` do nosso programa é igual a zero, ou não. Se for, então não devemos usar o método de Bhaskara e nosso programa deve avisar isso ao usuário.

Valores `bool` podem ser guardados em variáveis e podem ser utilizadas em expressões. Para criarmos expressões booleanas, existem operadores que servem para comparar operandos. Por exemplo:

```
>>> a = 10.01
>>> a == 10
False
>>> a > 10
True
>>> a != 10
True
>>> 'abc' < 'def'
True
>>>
```

A Tabela 10.1 mostra um resumo dos operadores que produzem resultados booleanos, também conhecidos por operadores relacionais. Note que eles podem ser utilizados para comparar valores de vários tipos diferentes como `int`, `float` ou string. Mas os tipos dos operandos têm que ser compatíveis. Não é possível comparar, por exemplo, um número e um string.

Existem também operadores lógicos que servem para combinar expressões booleanas. Por exemplo, se tivermos uma variável numérica `x` e gostaríamos

Operador	Significado	Exemplo	Resultado
==	Igual	10 == 10.1	False
		"abc"== "abc"	True
!=	Diferente	10 != 10.1	True
		"abc"!= "abc"	False
<	Menor	10.1 < 10	False
		"abc"< "def"	True
>	Maior	10.1 > 10	True
		"abc"> "def"	False
<=	Menor ou Igual	10.1 <= 10	False
		"abc"<= "abc"	True
>=	Maior ou Igual	10.1 >= 10	True
		"abc">= "def"	False

Tabela 10.1: Operados relacionais

de saber se ela está no intervalo $[0, 1]$. Para fazer tal verificação, precisamos, na verdade fazer duas comparações. A variável deve, ao mesmo tempo, ser menor ou igual a 1 e maior ou igual a 0. Note que na primeira expressão ambas as subexpressões são verdadeiras e portanto a expressão como um todo é verdadeira. Na segunda, a subexpressão da esquerda é verdadeira mas a da direita é falsa e portanto o resultado é falso.

```
>>> x = 0.5
>>> x >= 0.0 and x <= 1.0
True
>>> x = 2.0
>>> x >= 0.0 and x <= 1.0
False
>>>
```

Se, ao contrário do exemplo anterior, quiséssemos saber se o valor de `x` está fora do intervalo dado, precisaríamos verificar se ele é maior que 1 ou menor que 0. Ou seja, a expressão abaixo é verdadeira se pelo menos uma das subexpressões é verdadeira.

```
>>> x = 0.5
>>> x < 0.0 or x > 1.0
False
>>> x = 2.0
>>> x < 0.0 or x > 1.0
True
>>>
```

Nesse exemplo é impossível que as duas subexpressões sejam verdadeiras mas em outros casos isso é possível, o que faria com que o resultado da expressão toda seja verdadeiro pois pelo menos uma subexpressão deve ser verdadeira.

```
>>> x = 0.5
>>> y = 1.5
>>> x < 0.0 or y > 1.0
True
>>> x == 0.0 or x != y
True
>>>
```

Existe, ainda, o operador unário "not" cujo resultado é a negação do valor ao qual ele é aplicado. Por exemplo, se quisermos saber se o valor de `x`, no exemplo anterior, não está no intervalo $[0, 1]$ podemos utilizar a expressão que segue:

```
>>> x = 0.5
>>> not (x >= 0.0 and x <= 1.0)
False
>>> x = 2.0
>>> not (x >= 0.0 and x <= 1.0)
True
>>>
```

A Tabela 10.2 mostra o que chamamos "tabela verdade" para cada um dos operadores lógicos. Ou seja, para cada combinação possível dos operandos, ela mostra qual seria o resultado da expressão.

Tabela 10.2: Tabela verdade dos operadores lógicos

Operação	a	b	Resultado
a and b	False	False	False
	False	True	False
	True	False	False
	True	True	True
a or b	False	False	False
	False	True	True
	True	False	True
	True	True	True
not a	False	-	True
	True	-	False

10.1 O comando `if/else`

Agora que conhecemos os valores e expressões booleanos, podemos introduzir o comando de seleção `if`. Ele serve para modificar o fluxo de execução do nosso programa. Por exemplo, se o valor de Δ computado no método de Bhaskara for negativo, não queremos computar o valor das raízes, pois obteríamos um erro ao tentar calcular a raiz quadrada de um número negativo. O que queremos é mostrar apenas uma mensagem, dizendo que a equação não tem solução real.

O comando `if` pode ser entendido da seguinte maneira:

Programa 10.1 Como funciona o comando `if/else`

```
1 if <expressão booleana> :
2     comando executado se expressão for verdadeira
3     comando executado se expressão for verdadeira
4     comando executado se expressão for verdadeira
5 else :
6     comando executado se expressão for falsa
7     comando executado se expressão for falsa
```

Ele inicia com a palavra **if** que vem seguida de uma expressão cujo resultado deve ser do tipo `bool` seguida de ":". Se o resultado for verdadeiro, então os comandos que estiverem nas linhas seguintes são executados. Mas notem que para indicarmos quais são esses comandos, devemos colocá-los todos em um nível a mais de indentação. No exemplo acima, são três comandos executados, um após o outro, se a condição do `if` for `True`.

Em seguida, vem o comando `else`, que assim como o `if` deve ter um ":" no final da linha. O comando `else` deve estar no mesmo nível de indentação do `if` e abaixo dele vêm os comandos que serão executados se a expressão booleana der resultado `False`. Resumidamente podemos compreender o comando `if` como "se" e o comando `else` como "senão".

Então, podemos melhorar o nosso programa *bhaskara.py*, adicionando a verificação se a variável `delta` é negativa. Note que depois do comando `if` colocamos

um *print* que está no mesmo nível de indentação do `if` e dos comandos anteriores. Ou seja, não está "dentro" do comando `if` e será executado, qualquer que seja o valor da expressão booleana computada no `if`.

Programa 10.2 Uso do `if` no programa *bhaskara.py*

```
1  import math
2
3  a = float(input('Digite o valor de a: '))
4  b = float(input('Digite o valor de a: '))
5  c = float(input('Digite o valor de c: '))
6
7  delta = b ** 2 - 4 * a * c
8
9  # Verfica se delta é negativo
10 if delta < 0:
11     print('Essa equação não possui raízes reais')
12 else:
13     x1 = (-b + math.sqrt(delta)) / (2 * a)
14     x2 = (-b - math.sqrt(delta)) / (2 * a)
15
16     print('O valor da 1a raiz é {:.4f}'.format(x1))
17     print('O valor da 2a raiz é {:.4f}'.format(x2))
18
19
20 print('Final do programa')
```

O comando `else` deve sempre estar associado a um comando `if`. Ele não existe sozinho. Por outro lado, ele não é obrigatório, ou seja, podemos ter comandos `if` sem `else`.

Programa 10.3 Comando `if` sem `else`

```
1 if <expressão booleana> :
2     comando executado se expressão for verdadeira
3     comando executado se expressão for verdadeira
4     comando executado se expressão for verdadeira
```

Nesse caso, se a expressão booleana for verdadeira, os mesmos comandos que seguem o `if` são executados. mas se for falsa, nada é executado, ou melhor, o programa prossegue com o próximo comando depois do `if`, no mesmo nível de indentação. Um exemplo prático disso também pode ser mostrado no algoritmo de Bhaskara. Antes de executar os cálculos, nós vamos verificar se a equação realmente é quadrática, ou seja, se o valor da variável `a` é diferente de zero. Se ela não for quadrática, nosso programa deve apenas emitir um aviso e deve terminar, sem fazer o resto.

Programa 10.4 Uso do `if` sem `else` no programa *bhaskara.py*

```
1  import math
2  import sys
3
4  a = float(input('Digite o valor de a: '))
5  b = float(input('Digite o valor de a: '))
6  c = float(input('Digite o valor de c: '))
7
8  #verifica se é equação quadrática
9  if a == 0:
10     print('Essa equação não é quadrática.')
11     print('Terminando a execução')
12     sys.exit()
13
14 delta = b ** 2 - 4 * a * c
15
16 # Verfica se delta é negativo
17 if delta < 0:
18     print('Essa equação não possui raízes reais')
19 else:
20     x1 = (-b + math.sqrt(delta)) / (2 * a)
21     x2 = (-b - math.sqrt(delta)) / (2 * a)
22
23     print('O valor da 1a raiz é {:.4f}'.format(x1))
24     print('O valor da 2a raiz é {:.4f}'.format(x2))
25
26
27 print('Final do programa')
```

Veja que, após ler do teclado os valores dos coeficientes, nosso programa verifica se a variável `a` é igual a zero. Se for, ele emite uma mensagem e chama a função `sys.exit()` que termina a execução do programa imediatamente. Mas se não for, não existe uma alternativa no comando `if`. O programa simplesmente continua a sua execução no próximo comando, que é o cálculo do Δ.

10.1.1 Exercícios

1. Execute os programas 10.2 e 10.4 desta seção com valores diferentes e analise os resultados produzidos. Use pelo menos algum caso em que:

   a) a equação não tem solução real;

   b) a equação não é quadrática

10.2 Aplicação: método da bisseção

Embora de forma bastante precária, podemos agora implementar o método da bisseção. Por hora, temos que definir a função que desejamos usar com uma

expressão lambda, no início do programa. Ou seja, se quisermos mudar a função, temos que alterar o programa.

Vamos, também, inicializar os valores das variáveis que determinam o intervalo inicial e o erro que consideramos aceitável. Assim, o nosso programa inicia-se com os seguintes comandos:

Programa 10.5 Início do método da bisseção

```
1 import sys
2
3 f = lambda x: x ** 3 - x ** 2 - 13 * x + 8
4 a = -4.0
5 b = -3.0
6 erro = 0.001
```

Isso feito podemos começar as iterações para ir diminuindo o intervalo de busca da raiz. Ou seja, computamos o ponto médio entre `a` e `b`, verificamos o tamanho do intervalo, para decidir se podemos ou não parar e, se não podemos, alteramos o valor das variáveis para refletir o novo intervalo. A primeira iteração, então fica:

Programa 10.6 Uma iteração do método da bisseção

```
1 #iteração 1
2 c = (a+b)/2
3 if abs(( b - a ) / 2) < erro:
4     print('Achou raiz ', c, ' com erro ', (b-a)/2)
5     sys.exit();
6 if f(a) * f(c) < 0:
7     b = c
8 else:
9     a = c
```

Note que depois de executar esse trecho do programa, teremos, novamente que computar o valor de `c`, verificar o tamanho do intervalo etc. Isso significa que podemos começar a segunda iteração, que é exatamente igual à primeira. Ou seja, o resto do nosso programa é uma repetição, várias vezes, do código acima. Se quisermos que o nosso programa tenha dez iterações, repetimos dez vezes o mesmo trecho.

Ao executar o programa, os comandos vão sendo executados sequencialmente e quando o intervalo for menor do que o erro, a condição do primeiro `if` é executada e o programa termina. Isso pode acontecer na primeira, na segunda, ou qualquer uma das iterações, dependendo da função, do intervalo, e do valor escolhido para o erro. Por exemplo, usando os valores acima, o programa termina na décima iteração. Se a tolerância for de 0.01, o programa terminaria na sétima iteração.

Mas também é possível que após a décima iteração, o tamanho do intervalo continue maior do que a tolerância que definimos. Isso acontece, por exemplo, se definirmos a variável `erro` com um valor 0,0001. Para esses casos, precisamos colocar, no final do nosso programa, um comando para avisar ao usuário qual foi a solução encontrada e qual o erro obtido. Adicionamos, então, o seguinte comando:

Programa 10.7 Finalizando o método da bisseção

```
1 print('Valor calculado ', c, ' com erro ', (b-a)/2)
```

10.3 O comando `if/elif/else`

O comando `if` admite uma outra forma, que permite que várias expressões booleanas sejam testadas em sequência. Quando uma delas for verdadeira, os comandos correspondentes são executados e todas as outras expressões booleanas serão ignoradas. Caso nenhuma das expressões for verdadeira, podemos ter, opcionalmente o comando `else`.

Programa 10.8 Como funciona o comando `if`/`elif`/`else`

```
1 if <expressão 1> :
2     comando executado se expressão 1 for verdadeira
3 elif <expressão 2> :
4     comando executado se expressão 2 for verdadeira
5 elif <expressão 3> :
6     comando executado se expressão 3 for verdadeira
7 else :
8     comando executado se todas expressões forem falsas
```

Podemos ter quantas expressões quisermos. A primeira verdadeira determina os comandos que serão executados e as demais, abaixo dela, não serão verificadas e seus comandos não serão executados. O comando `else` pode aparecer ou não.

Se você tentou implementar o programa da bisseção, no final da seção anterior, poderia ter usado esse tipo de comando `if`. Depois de computar o valor médio entre `a` e `b`, é preciso escolher qual subintervalo utilizar na próxima iteração. Para isso, verificamos o valor de $f(a) * f(c)$. Se esse valor for negativo, iremos utilizar o intervalo entre `a` e `c`. Se for positivo, o intervalo entre `c` e `b`. E se for zero, isso significa que $f(c)$ é exatamento zero e portanto `c` tem o valor da raiz da função. Traduzindo isso para a linguagem Python, teríamos algo como:

Programa 10.9 Como funciona o comando `if/elif/else`

```
1 c = (a+b)/2
2 if f(a) * f(c) < 0:
3     b = c
4 elif f(a) * f(c) > 0:
5     a = c
6 else:
7     print('O valor exato da raiz é ', c)
8     sys.exit()
```

10.3.1 Exercícios

1. Reimplemente o algoritmo da bisseção com, no máximo, 10 iterações. Use o comando `if/elif/else` para verificar qual é o próximo subintervalo que o programa deve usar para procurar a raiz, conforme explicado nesta seção.

2. Reimplemente o algoritmo de Newton-Raphson com, no máximo, 10 iterações. A cada iteração, use o comando `if` para verificar se o erro está abaixo de 10^{-7}. Se estiver, termine imediatamente o programa, exibindo qual é a raiz e qual é o erro.

10.4 Comandos aninhados

O comando `if`, assim como outros comandos que veremos adiante, possui "dentro" dele outros comandos. Nos exemplos que vimos há pouco, utilizamos comandos simples como atribuições e o `print` mas podemos utilizar quaisquer comandos, inclusive o próprio `if`.

Tomando novamente como exemplo o método de Bhaskara (Programa 10.4) podemos reescrever a sua implementação da seguinte forma:

Programa 10.10 `ifs` aninhados no programa *bhaskara.py*

```
1  import math
2  import sys
3
4  a = float(input('Digite o valor de a: '))
5  b = float(input('Digite o valor de b: '))
6  c = float(input('Digite o valor de c: '))
7
8  #verifica se é equação quadrática
9  if a == 0:
10     print('Essa equação não é quadrática.')
11 else:
12     delta = b ** 2 - 4 * a * c
13
14     # Verfica se delta é negativo
15     if delta < 0:
16         print('Essa equação não possui raízes reais')
17     else:
18         x1 = (-b + math.sqrt(delta)) / (2 * a)
19         x2 = (-b - math.sqrt(delta)) / (2 * a)
20
21         print('A 1a raiz é {:.4f}'.format(x1))
22         print('A 2a raiz é {:.4f}'.format(x2))
23
24 print('Final do programa')
```

Neste programa, vemos que a condição do primeiro `if` (linha 9) é executada e, se for verdadeira, a execução segue para os dois `print` (linhas 10 e 11). Se a condição for falsa, a execução vai direto para os comandos que estão dentro do `else` da linha 12. Só que dentro desse `else` temos o cálculo do valor de `delta` e depois um outro comando `if/else`. Nesse caso, a condição da linha 16 é verificada e se for verdadeira executa-se o `print` da linha 17. Se for falsa, são os comandos das linhas 19 a 23 que serão executados.

Note que a indentação é que determina qual comando está aninhado a outro comando. O comando `if/else` que começa na linha 16 está dentro do `else` da linha 12. Por isso, está um nível de indentação à direita dele. Os comandos que estão dentro desse segundo `if`, ou seja, linhas 17 e 19 a 23, devem então estar um nível mais à direita ainda. É dessa forma que o interpretador Python identifica a estrutura de comandos do nosso programa.

10.4.1 Exercícios

1. Crie um programa que permita que o usuário informe um número qualquer e então o programa deve exibir a raiz quadrada deste número. Caso o número informado pelo usuário seja um número negativo, utilize o módulo deste número para calcular a raiz quadrada. Exemplos:

   (a) Caso o número informado pelo usuário seja 16, o resultado deve ser 4.

   (b) Caso o número informado pelo usuário seja -36, o resultado deve ser 6.

2. Crie um programa que permita que o usuário informe duas notas e com base nelas, calcule a média, mostre sua nota final e com base nesta nota final, exiba se ele foi aprovado, se ficou de recuperação ou se foi reprovado. Para ser aprovado o aluno tem que obter uma nota maior ou igual a 7. Para ser reprovado o aluno tem que obter uma nota menor que 5. Exemplos:

   (a) Se a primeira nota informada for 8 e a segunda for 7,5, o resultado deve ser "Nota Final: 7.75 | Aprovado".

   (b) Se a primeira nota informada for 7,5 e a segunda for 5,50, o resultado deve ser "Nota Final: 6.50 | Recuperação".

   (c) Se a primeira nota informada for 2,5 e a segunda for 6,5, o resultado deve ser "Nota Final: 4.50 | Reprovado".

3. Escreva um programa que gere aleatoriamente um número entre 0 e 100. Depois, o programa deve dar até 10 chances para o usuário adivinhar qual é o numero secreto. A cada palpite, o programa diz ao usuário se seu palpite é maior, menor, ou se ele acertou o valor. Se o usuário acertar o valor, o programa termina. Para gerar um número aleatório use a função *random.randint*.

4. Escreva um programa que lê 3 valores que representam os lados de um triângulo. O programa deve dizer se eles correspondem a um triângulo equilátero, isósceles ou escaleno ou, ainda se não correspondem a um triângulo.

5. Faça um programa que leia o sexo e a altura de uma pessoa e calcule o seu peso ideal, utilizando as seguintes fórmulas:

   para homem: (72.7 x h) – 58

   para mulher: (62.1 x h) – 44.7

6. Escreva um programa que lê o valor do salário de um trabalhador e calcula o valor do imposto de renda a ser recolhido na fonte, de acordo com a tabela da Receita Federal.

7. Escreva um programa que verifica se um determinado ano é ou não bissexto.