Capítulo 9

Programando de verdade

Até agora, o processo que usamos para resolver nossos problemas foi bastante manual. Apenas utilizamos o interpretador Python para fazer umas contas e guardar alguns valores em variáveis. O que queremos é uma forma de fornecer os comandos para o interpretador e deixar por conta dele a execução.

Isso é muito fácil de fazer. Nós precisamos colocar a sequência de comandos que deve ser executada pelo interpretador em um arquivo texto e depois instruir o interpretador a executar aquele arquivo, ou mais precisamente, os comandos que estão listados naquele arquivo.

Por exemplo, suponha que colocamos os comandos do algoritmos de Bhaskara em um arquivo com nome *bhaskara.py*. Costumamos usar essa extensão para os programas Python, para, quando acharmos um arquivo com essa extensão, sabermos do que se trata. Para executar os comandos do arquivo devemos, no *prompt* do Windows ou no console do Linux, invocar o seguinte comando:

```
>> python bhaskara.py
```

A partir daí, o interpretador Python é carregado e passa a executar os comandos que estão no arquivo. Finalizada a execução dos comandos do arquivo, o interpretador também termina sua execução e o controle volta para o console do usuário. Para quem usa algum dos ambientes de programação mencionados no Apêndice A, não é necessário executar um console para invocar o interpretador e executar o programa. Dentro do próprio ambiente de programação existe uma opção para executar o programa.

É importante notar que, assim como acontece quando interagimos diretamente com o interpretador Python, ao executar o programa do arquivo é possível que exista algum erro nos comandos ali armazenados. Nesse caso, o interpretador pára a execução do programa, avisa ao programador qual o erro encontrado e qual a linha do programa onde ele ocorreu. Por exemplo, se esquecermos de incluir o comando `import math` no nosso programa, teremos o seguinte resultado:

```
>> python bhaskara.py
Traceback (most recent call last):
  File "bhaskara.py", line 8, in <module>
    x1 = (-b + math.sqrt(delta)) / (2 * a)
NameError: name 'math' is not defined

>>
```

9.1 Comando de saída

Vamos então escrever no arquivo *bhaskara.py* os comandos que usamos para computar as raízes da nossa função $f(x) = 308x^2 + 113x - 1033$. Programa 9.1.

Programa 9.1 Versão inicial do método de Bhaskara

```
1  import math
2
3  a = 308
4  b = 113
5  c = -1033
6
7  delta = b ** 2 - 4 * a * c
8
9  x1 = (-b + math.sqrt(delta)) / (2 * a)
10 x2 = (-b - math.sqrt(delta)) / (2 * a)
11
12 x1
13 x2
```

Porém, ao executarmos esse programa, conforme definimos anteriormente, nada acontece, ou seja, os valores de `x1` e `x2` não são apresentados, como gostaríamos que acontecesse. Isso ocorre porque, ao executarmos o programa armazenado em um arquivo precisamos incluir comandos que explicitamente indiquem que queremos apresentar algum valor "na saída" do programa, ou seja, no console. Para fazermos isso vamos usar a função *print*.

Essa função recebe zero, um ou mais parâmetros e como resultado, mostra o valor de cada um dos parâmetros, todos seguidos em uma única linha. Se for chamada sem nenhum parâmetro, a função simplesmente escreve uma "linha vazia", ou seja, não escreve nada mas o próximo *print* do programa ira ocupar uma nova linha. Então, se trocarmos os dois últimos comandos do Programa 9.1 por:

```
1 print(x1)
2 print(x2)
```

teremos como resultado os valores armazenados nas variáveis `x1` e `x2`.

```
>> python bhaskara.py
1.6570874169509975
-2.0239705338341145
>>
```

Podemos melhorar um pouco essa saída, fornecida por nosso programa. Já que a função *print* permite que passemos vários valores seguidos, podemos deixar a saída mais bonita ou mais explicativa com os comandos:

```
1 print("O valor da 1a raiz é ", x1)
2 print("O valor da 2a raiz é ", x2)
```

Esses dois comandos produzem como resultado do programa a seguinte saída:

```
>> python bhaskara.py
O valor da 1a raiz é 1.6570874169509975
O valor da 2a raiz é -2.0239705338341145
>>
```

Uma outra forma de obter o mesmo resultado é concatenando vários objetos para formar um único string dentro do *print*. Nesse caso, para fazer a concatenação precisamos converter todos os objetos para string. O programa, então, ficaria como mostrado a seguir.

Programa 9.2 Segunda versão do método de Bhaskara

```
1  import math
2
3  a = 308
4  b = 113
5  c = -1033
6
7  delta = b ** 2 - 4 * a * c
8
9  x1 = (-b + math.sqrt(delta)) / (2 * a)
10 x2 = (-b - math.sqrt(delta)) / (2 * a)
11
12 print("O valor da 1a raiz é " + str(x1))
13 print("O valor da 2a raiz é " + str(x2))
```

9.1.1 Exercícios

1. Modifique o arquivo *bhaskara.py* para que ele resolva as outras equações que vimos no capítulo anterior:

   a) $20x^2 + 214x + 572,45 = 0$

   b) $211x^2 - 14x + 103 = 0$

   c) $211x^2 - 103 = 0$

   d) $211x^2 - 14x = 0$

   e) $211x^2 = 0$

9.2 Comando de entrada

Se você resolveu os exercícios da seção anterior, deve ter notado um grande inconveniente no nosso programa. Acontece que para cada equação que queremos resolver precisamos alterar os primeiros comandos do arquivo *bhaskara.py*, nos quais os coeficientes são atribuídos às variáveis `a`, `b` e `c`. Seria mais interessante se tivéssemos um único programa para resolver qualquer equação do segundo grau, sem que precisássemos alterá-lo, para cada equação distinta.

A função *input()* é utilizada para fazer com que o usuário interaja com o seu programa, fornecendo a ele um valor. Esse valor pode, então, ser utilizado nos cálculos dentro do programa. Crie um arquivo com os seguintes comandos e verifique o seu resultado.

Programa 9.3 Uso da função *input*

```
1 s = input()
2 print(s)
```

O que acontece ao executar a primeira linha, é que o programa pára sua execução e fica esperando você digitar um texto qualquer, até terminar com um `<enter>`. O valor que você digitar é atribuído à variável `s` e depois exibida de volta para você, por meio da função *print*.

```
>> python teste_input.py
Vou digitar um texto
Vou digitar um texto
>>
```

A primeira ocorrência de "Vou digitar um texto", acima, corresponde ao texto que vai aparecendo à medida que você digita, até pressionar a tecla `<enter>`. A segunda ocorrência corresponde à execução do *print* com o valor da variável `s`.

Para facilitar um pouco, é possível associar à função *input*, uma mensagem que aparece para o usuário, para que ele saiba que o programa está esperando a

digitação de um valor. No exemplo abaixo, é mostrada uma mensagem "Digite seu nome" e então o programa espera a digitação de um texto, que é atribuído à variável `s`.

Programa 9.4 Uso da função *input*

```
1 s = input("Digite seu nome: ")
2 print(s)
```

Veja a execução:

```
>> python teste_input.py
Digite seu nome: José da Silva
José da Silva
>>
```

Podemos substituir agora os primeiros comandos do programa *bhaskara.py* pelo uso da função *input*. Só precisamos tomar cuidado pois os valores atribuídos às variáveis `a`, `b` e `c` devem ser do tipo `float` para que elas possam ser usadas em cálculos matemáticos mas a função *input* retorna um string. Então, precisamos converter os valores digitados para um número, antes de atribuí-los às variáveis.

Programa 9.5 Programa *bhaskara.py* com a função *input*

```
1  import math
2
3  s = input('Digite o valor de a: ')
4  a = float(s)
5  s = input('Digite o valor de b: ')
6  b = float(s)
7  s = input('Digite o valor de c: ')
8  c = float(s)
9
10
11 delta = b ** 2 - 4 * a * c
12
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 ", x1)
17 print("O valor da 2a raiz é ", x2)
```

Executando esse programa, podemos digitar quaisque valores para os coeficientes da equação e, assim, não precisamos alterar o programa a cada execução. Se quisermos, por exemplo resolver a equação $20x^2 + 214x + 572,45 = 0$.

```
>> python bhaskara.py
Digite o valor de a: 20
Digite o valor de b: 214
Digite o valor de c: 572.45
O valor da 1a raiz é -5.35
O valor da 2a raiz é -5.35
>>
```

9.2.1 Exercícios

1. Use o programa desta seção para resolver as outras equações quadráticas que vimos anteriormente. O que acontece se a equação que você escolheu não for quadrática ($a = 0$) ou não possuir solução real ($\Delta < 0$)?

2. O que acontece se você digitar, ao ser solicitado o valor de uma das variáveis, alguma coisa que não seja um número, por exemplo: "abcd"?

9.3 Indentação e comentários

Você deve ter notado que os comandos no nosso programa (arquivo *.py*) possui alguma linha em branco, que servem para agrupar alguns comandos e separar outros. Por exemplo, o `import` está separado dos comandos de leitura das variáveis, que estão agrupados e por sua vez separados do cálculo do Δ e assim por diante. Nesse sentido, não existe restrição quanto ao uso de linhas em branco. Por outro lado, todas as linhas precisam começar sem nenhuma indentação, ou seja, na primeira coluna.

Veremos mais adiante que o Python usa a indentação das linhas para identificar a estrutura do programa. Além disso, mais adiante, quando for usar indentação, não misture espaços em branco com "tabs". Isso confunde o interpretador, que vai reclamar e não vai executar o seu programa.

Um outro aspecto importante na organização dos nossos programas é o uso de comentários. Um comentário dentro do seu programa não é executado pelo interpretador. Ele serve somente para que você documente o seu programa e quando você ou outra pessoa precisar ler e entender o que ele faz, a tarefa seja facilitada. Uma linha de comentário em um programa Python inicia com o símbolo "#" e a linha inteira depois dele é ignorada.

Não existe qualquer requisito de indentação em relação aos comentários, que podem, também, aparecer no final de uma linha, depois dos comandos. Como exemplo, veja em seguida como podemos adicionar comentários no nosso programa de resolução de equações quadráticas.

Programa 9.6 Comentários no programa *bhaskara.py*

```
1  # importa as funções matemáticas
2  import math
3
4  # inicializa as variáveis, lendo do teclado o valor
5  # dos coeficientes
6  s = input('Digite o valor de a: ')
7  a = float(s)
8  s = input('Digite o valor de b: ')
9  b = float(s)
10 s = input('Digite o valor de c: ')
11 c = float(s)
12
13 # aplica o método de Bhaskara
14 delta = b ** 2 - 4 * a * c # calcula o delta
15
16 # computa o valor das raízes
17 x1 = (-b + math.sqrt(delta)) / (2 * a)
18 x2 = (-b - math.sqrt(delta)) / (2 * a)
19
20 # exibe o resultado
21 print("O valor da 1a raiz é ", x1)
22 print("O valor da 2a raiz é ", x2)
```

Mais um detalhe importante em relação à formatação do nosso programa: se tivermos uma linha muito grande, ou se por qualquer outro motivo quisermos quebrar um comando em mais do que uma linha podemos usar o caractere \ no final da linha para indicar que o comando continua na linha seguinte. Na linha subsequente, não há restrição quanto à indentação pois essa linha é apenas uma continuação da linha anterior. Por exemplo, vamos quebrar as duas atribuições de `x1` e `x2`, em dois pontos diferentes do comando.

Programa 9.7 Quebrando uma linha do programa

```
1  # computa o valor das raízes
2  x1 = (-b + math.sqrt(delta)) / \
3       (2 * a)
4  x2 = (-b - math.sqrt(delta)) \
5       / (2 * a)
```

9.4 Exercícios

1. Faça um programa que leia duas notas de um aluno, calcule a média simples do aluno, e depois mostre como saída a média calculada.

2. Faça um programa que leia três notas de um aluno, onde a primeira e segunda nota possuem peso um e a terceira nota possui peso dois. Calcule a média ponderada destas notas e depois mostre na saída o resultado

3. Faça um programa que leia uma temperatura fornecida em graus Celsius e converta para graus Fahrenheit, exibindo o resultado na saída.

4. Faça um programa que leia o valor da hora de trabalho (em reais) e número de horas trabalhadas no mês, e exiba na tela o valor a ser pago ao funcionário, adicionando 10% sobre o valor calculado.

5. O valor do seno de x pode ser calculado pela série de Taylor, dada por:

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}...$$

   Faça um programa que leia o valor de x e compute o valor do seno usando os 5 primeiros termos desta série. Mostre o valor calculado e o valor fornecido pela função padrão do Python, $sin(x)$.

6. Escreva um programa para computar uma raiz da função $f(x) = x^3 - x^2 - 13x + 8$ usando 10 iterações do método de Newton-Raphson. Use um comando de entrada para ler o chute inicial.

7. Você conseguiria escrever um programa para implementar o método da bisseção, com 10 iterações? Qual seria o maior problema para fazer isso?

9.5 Formatação da saída

O comando `print` é relativamente limitado. Ele simplesmente mostra na saída os valores que são passados como parâmetros. Muitas vezes queremos dar uma forma mais agradável de visualização às saídas produzidas por nossos programas. É o caso, por exemplo, do programa *bhaskara.py* que poderia produzir uma saída com apenas 4 casas decimais. Ou então, uma saída que represente valores monetários, poderia, também ajustar a saída para esse padrão (R$10.00 por exemplo).

A linguagem Python fornece uma função `format` que serve para formatar um string, da maneira que desejarmos. Esse string pode, então, ser usado em um comando `print`. Para entender, vamos tomar como exemplo o programa para o método de Bhaskara, no qual queremos apresentar a como saída as duas raízes calculadas. Com o `format` podemos usar o seguinte comando:

```
1 print("A 1a raiz é {} e a 2a é {}".format(x1,x2))
```

O string usado na formatação é o que está antes do ponto (em vermelho), na chamada da função `format` e os parâmetros são os valores de `x1` e `x2`. Note que no string de formatação existem dois "campos" representados por `{}`. Esses campos vão ser substituídos pelos valores dos parâmetros passados para a função `format`, ou seja, os valores de `x1` e `x2`. Então, o valor do string que vai ser mostrado pelo comando `print` é:

```
A 1a raiz é 1.6570874169509975 e a 2a é -2.0239705338341145
```

Mas esse resultado poderíamos ter obtido apenas com algumas concatenações de strings. Porém, o string de formatação permite que adicionemos, dentro dos campos substituíveis, algumas instruções de como os parâmetros vão ser formatados. No exemplo a seguir, formatamos as saídas com quatro casas decimais. O ":" é obrigatório e indica o ínicio da formatação. A letra `f` indica que estamos formatando um valor `float` e o ".4" indica exatamente que queremos quatro casas decimais.

```
1 "A 1a raiz é {:.4f} e a 2a é {:.4f}".format(x1,x2)
```

O string produzido é mostrada a seguir. Note que na formatação os números não são apenas truncados na quarta casa decimal mas sim arredondados.

```
A 1a raiz é 1.6571 e a 2a é -2.0240
```

Podemos, também, estabelecer qual o tamanho total que cada campo vai ocupar no string final. Por exemplo, vamos estabelecer que queremos que cada valor ocupe nove espaços na saída, sendo quatro deles casas decimais. Além disso, o valor de `x1` vai ser ajustado à esquerda desses nove caracteres, por isso usamos o sinal "<". O valor de `x2` vai ser ajustado à direita, por isso usamos o sinal ">". Se quisermos centralizar o valor nesse espaço que determinamos, podemos usar "^".

```
1 "A 1a raiz é {:<9.4f} e a 2a é {:>9.4f}".format(x1,x2)
2 "A 1a raiz é {:^9.4f} e a 2a é {:^9.4f}".format(x1,x2)
```

As saídas produzidas nesses dois casos são:

```
A 1a raiz é 1.6571    e a 2a é   -2.0240
A 1a raiz é  1.6571   e a 2a é  -2.0240
```

Podemos usar as mesmas regras para formatar números inteiros, usando a letra "`d`" em vez do '`f`". Além disso, para inteiros não é permitido que se determine o número de casas decimais. O exemplo a seguir mostra vários números que são formatados todos com quatro espaços. Devemos reparar que quando um número não cabe no espaço reservado, como o último do exemplo que segue, não há nenhum truncamento no número. Ele simplesmente extrapola o espaço que deveria ocupar.

```
1 '{:>4d}{:>4d}{:>4d}{:>4d}'.format(12, 102, 1002, 10002)
```

No quadro a seguir vemos cada um dos números formatados e, na linha seguinte, uma indicação de onde termina cada um dos campos.

```
 12 102100210002
  |   |   |   |
```

A função `format` é bem mais complexa do que explicamos aqui. O que vimos aqui são algumas funcionalidades básicas mas que vão nos ajudar no restante deste livro. Para ter uma visão completa dessa funçao, o leitor deve consultar a documentação de Python em `https://docs.python.org/3/library/string.html#formatstrings`.

Uma outra forma muito comum de formatar a saída de um programa é utilizando a sequência "\n" dentro de um string em um comando `print`. Essa sequência faz com que haja uma quebra de linha, ou seja, a parte que vem depois dela no string é apresentada em uma nova linha na saída. Por exemplo,

```
1 print('x1 = {:9.4f}\nx2={:9.4f}'.format(x1,x2))
```

vai produzir a saída

```
x1 =    1.6571
x2 =   -2.0240
```

9.5.1 Exercícios

Repita os exercícios da seção anterior, formatando de maneira adequada as saídas produzidas pelos seus programas.