Este capítulo aborda algumas técnicas para encontrar a causa de bugs no seu programa para ajudá-lo a corrigi-los mais rápido e com menos esforço.Quanto mais cedo identificados,mais rápido e fácil de ser visualizado ele sera.

•Criar exceções

O Python gera uma exceção sempre que tenta executar um programa com algum erro, você pode criar sua própria exceção.Criar uma exceção é uma maneira para que o programa pare de executar o código em uma certa função e mova a execução do programa para a instrução except. As exceções são feitas com uma instrução de raise.Ela é feita da seguinte maneira:

- A palavra-chave raise;
- Uma chamada para a função Exception ();
- Uma seqüência de caracteres com uma mensagem de erro útil passada para a função Exception ().

Se não houver instruções try e except que mostram a instrução raise que mostrou a exceção, o programa simplesmente trava e exibe a mensagem de erro onde esta aexceção.

•Rastreamneto (Traceback) como uma string

Quando se econtra um erro em Python,ele possui um identificador de erro chamado rastreamento(Traceback) onde mostra, a linha ,coluna e onde a função foi chamada que levou ao erro. O traceback é exibido pelo Python sempre que uma exceção levantada não é tratada.

•Afirmação

A afirmação é uma função que diz se o programa executado esta fazendo um bugg muito óbvil. Essas verificações são executadas por uma declaração assert.A instrução de assert consiste em: -Palavra-chave assert; -Uma condição ,que indica se é True ou False; -Uma vírgula; -Uma string a ser exibida quando a condição for False.

•Desabilitando Asserções

As asserções podem ser desativadas passando a opção -O quando o Python é executado. Isso é bom para quando você terminar de escrever e testar seu programa e não quer que ele seja abrandado. As afirmações são para o desenvolvimento, não para o produto final.

•Logging

Logging é uma das melhores maneiras de entender o que está acontecendo em seu programa e a ordem de cada acontecimento. O módulo de log do Python facilita a criação de um registro de mensagens personalizadas que você escreve.

As mensagens de log descrevem quando a função é chamada em seu registro e lista as variáveis especificadas.Mas se faltar uma mensagem de log ,indica a falta de uma parte do código que foi ignorada ou não executada.

•Não usar debugg com print ()

Após o termino do debugg se utilizar o print, muito tempo sera gasto removendo chamadas de impressão de seu código para cada mensagem de log. Você pode até mesmo remover acidentalmente algumas chamadas print () que estavam sendo usadas para mensagens nonlog.

•Níveis de logging

Os níveis de logging fornecem uma maneira de categorizar suas mensagens por importância de log: - logging.debug() = usado para identificar bug em pequenos detalhes; - logging.info() = confirmar que as coisas estão funcionando em seu ponto no programa; - logging.warning() = identificar pequenos problemas no programa,que pode ocasionar problema sna execução futura. - logging.error() = grava um erro que impede um programa de executar algo; - logging.critical() = identifica um erro de nível mais alto que pode fazer que o programa nao execute o código interalmente.

•IDLE's Debugger

Executa o programa linha por linha,identificando e avisando quando continuar a execução e se possivel.Ele vai executando uma por uma,ate acabar a "vida" do programa,extremamente impotante para identificar erros.

Go = executa o programa ate o fim ,até atingir um ponto de interrupção;
Step = clicar no botão step executará a próxima linha de código;
Out = clicar no botão Out fará com que execute linhas de código em velocidade máxima até que ele retorna da função atual.
Quit = irá sair do programa imediatamente;

•Breakpoints

Um breakpoint pode ser definido em uma linha específica de código e força o código a pausar a execução do programa ate que ela atinja essa linha.

Para completar o que foi aprendido nesse capitulo,fizemos o projeto proposto pelo livro:
PROJETO:For practice, write a program that does the following.
Debugging Coin Toss
The following program is meant to be a simple coin toss guessing game. The player gets two guesses (it's an easy game). However, the program has several

bugs in it. Run through the program a few times to find the bugs that keep the program from working correctly.

```
    import random
guess = ”
while guess not in (’heads’, ’tails’):
print(’Guess the coin toss! Enter heads or tails:’)
guess = input()
toss = random.randint(0, 1)  0 is tails, 1 is heads
if toss == guess:
print(’You got it!’)
else:
print(’Nope! Guess again!’)
guesss = input()
if toss == guess:
print(’You got it!’)
else:
print(’Nope. You are really bad at this game.’)
```

RESOLUÇÃO:

```
    import random
guess = ”
while guess not in (int(1), int(0)):
print(’Guess the coin toss! Enter 1 for heads or 0 for tails:’)
guess = int(input())
toss = random.randint(0, 1)  0 is tails, 1 is heads
if toss == guess:
print(’You got it!’)
else:
print(’Nope! Guess again!’)
guess = int(input())
if toss == guess:
print(’You got it!’)
else:
print(’Nope. You are really bad at this game.’)
```