python™

Rafael Vieira Coelho

rafaelvc2@gmail.com

PARTE 1

11 - Depuração e Testes

1.2 O que é depuração (debugging)?

- Programar é um processo complicado e, como é feito por seres humanos, frequentemente conduz a erros.

- Por mero capricho, erros em programas são chamados de bugs e o processo de encontrá-los e corrigi-los é chamado de depuração (debugging).

- Três tipos de erro podem acontecer em um programa:

  1. **Erros de sintaxe**

  2. **Erros em tempo de execução (runtime errors)**

  3. **Erros de semântica (erros de lógica)**

python™

1.2.1 Erros de sintaxe

- O interpretador do Python só executa um programa se ele estiver sintaticamente correto; caso contrário, o processo falha e retorna uma mensagem de erro.
- Sintaxe se refere à estrutura de um programa e às regras sobre esta estrutura.
- Por exemplo, em português, uma frase deve começar com uma letra maiúscula e terminar com um ponto.
- Se o seu programa tiver um único erro de sintaxe em algum lugar, o interpretador Python vai exibir uma mensagem de erro e vai terminar - e o programa não vai rodar.

python™

1.2.2. Erros em tempo de execução (runtime errors)

- Erro em tempo de execução é chamado assim pois só aparece quando você roda o programa.

- Esses erros são também conhecidos como exceções, porque normalmente indicam que alguma coisa excepcional (e ruim) aconteceu.

1.2.3 Erros de semântica (ou de lógica)

- Mesmo que o seu programa tenha um erro de semântica, ele vai rodar com sucesso, no sentido de que o computador não vai gerar nenhuma mensagem de erro.

- Só que o programa não vai fazer a coisa certa, vai fazer alguma outra coisa. Especificamente, aquilo que você tiver dito para ele fazer (o computador trabalha assim: seguindo ordens).

- O problema é que o programa que você escreveu não é aquele que você queria escrever. O significado do programa (sua semântica ou lógica) está errado.

1.2.4 Depuração experimental (*debugging*)

- Uma das habilidades mais importantes que você vai adquirir é a de depurar.
- De certa maneira, a depuração é como um trabalho de detetive. Você se depara com pistas, e tem que deduzir os processos e eventos que levaram aos resultados que aparecem.
- Uma vez que você tem uma ideia do que está errado, você modifica o seu programa e tenta de novo.

Como Sherlock Holmes mostrou: "Quando você tiver eliminado o impossível, aquilo que restou, ainda que improvável, deve ser a verdade." (Arthur Conan Doyle, O signo dos quatro).

python™

Tipos de Erro

Diferentes tipos de erros podem acontecer em um programa, e é útil distinguir entre eles para os localizar mais rapidamente:

• Erros de sintaxe são produzidos quando o interpretador está traduzindo o código fonte em bytecode. Estes erros indicam que existe algo errado com a sintaxe do programa.

Ex: Omitir o sinal de dois pontos (:) no final de uma declaração def produz a mensagem: SintaxError: invalid sintax.

Erros de sintaxe são geralmente fáceis de corrigir, bastando apenas que você descubra onde eles estão. Infelizmente, as mensagens de erro geralmente ajudam pouco.

As mensagens mais comuns são:
- SyntaxError: invalid syntax (Erro de sintaxe: sintaxe inválida)
- SyntaxError: invalid token (Erro de sintaxe: objeto inválido)

Tipos de Erro

• Erros de tempo de execução são produzidos se algo de errado acontece enquanto o programa está em execução.

A maioria das mensagens de tempo de execução incluem informação sobre onde o erro ocorreu e que função estava em execução.

Tipos de Erro

• Erros de semântica, chamado por alguns autores de erros de lógica, são problemas com um programa que compila e executa mas não tem o resultado esperado.

Ex: Uma expressão pode não ser avaliada da forma que você espera, produzindo um resultado inesperado.

Depuração

O primeiro passo na depuração é descobrir com que tipo de erro você está lidando.

A mensagem de erro diz a você onde, no código do programa, ocorreu o problema.

Na verdade, ela diz a você onde o Python encontrou o problema, não necessariamente onde o erro está.

Muitas vezes, o erro é anterior à localização da mensagem de erro, geralmente na linha precedente.

Como evitar erros de sintaxe?

1. Certifique-se que você não está utilizando uma palavra reservada de Python para um nome de variável.

2. Verifique a existência do sinal de dois pontos no final do cabeçalho de cada declaração composta, incluindo as declarações for, while, if, e def.

3. Verifique se a endentação está consistente. Você pode endentar com espaços ou com tabulações, mas é melhor não misturá-los. Cada nível deve ser aninhado com a mesma quantidade.

4. Assegure-se de que cada string no código tenha as aspas correspondentes.

Como evitar erros de sintaxe?

5. Se você tem strings de multilinhas criadas usando três aspas (simples ou duplas), assegure-se de que você terminou a string apropriadamente. Uma string terminada de forma inapropriada ou não terminada pode gerar um erro de invalid token (objeto inválido) no final do seu programa.

6. Um conjunto de parênteses, colchetes ou chaves não fechados corretamente faz com que o Python continue com a próxima linha como parte da declaração anterior. Geralmente, um erro ocorre quase imediatamente na linha seguinte.

7. Verique o clássico = ao invés de == dentro de uma condicional.

Como evitar erros de tempo de execução?

Se o programa travar e não fizer nada?

Se existe um laço em particular aonde você suspeita estar o problema, adicione uma declaração print imediatamente antes do laço que diga “entrando no laço” e uma outra imediatamente depois que diga “saindo do laço”.

Execute o programa. Se você receber a primeira mensagem e não a segunda, você tem um laço infinito.

Como evitar erros de tempo de execução?

Se o programa travar e não fizer nada?

• Na maioria das vezes, uma recursão infinita fará com que o programa execute por um tempo e então ele produzirá um erro do tipo Runtime Error: Maximum recursion depth exceeded (Erro de tempo de execução: Excedida a profundidade máxima de recursão).

• Se nenhum destes passos funcionar, comece a testar outros laços ou métodos ou funções recursivas.

Como evitar erros de tempo de execução?

Se algo vai errado durante a execução, o Python exibe uma mensagem que inclui o nome da exceção, a linha do código do programa onde o problema ocorre, e dados para investigação do erro, onde é descrita a pilha de execução de funções e métodos.

Tais dados identificam a função que está sendo executada no momento, e então a função que a invocou, e então a função que invocou aquela, e assim por diante (a pilha de execução de funções).

Ele também inclui o número da linha no respectivo arquivo onde cada uma dessas chamadas ocorre.

O primeiro passo é examinar o lugar no programa onde ocorreu o erro e tentar descobrir o que aconteceu.

Tipos Comuns de Erros

- **NameError (Erro de Nome):** Você está tentando utilizar uma variável que não existe no ambiente atual. Lembre-se que variáveis locais são locais. Você não pode referenciá-la fora da função onde ela foi definida.

- **TypeError (Erro de Tipo):** Existem várias causas possíveis:

  - Você está tentando utilizar um valor de forma imprópria. Exemplo: indexando uma string, lista ou tupla com alguma coisa que não é um inteiro.

  - Há uma incompatibilidade entre os itens em um formato de string e os itens passados para conversão. Isto pode acontecer se o número de itens não for igual ou se uma conversão inválida é chamada. Por exemplo: Passar uma string para a formatação de conversão %f.

  - Você está passando o número errado ou tipo de argumentos para uma função ou método.

Tipos Comuns de Erros

- **KeyError (Erro de Chave):** Você está tentando acessar um elemento de um dicionário utilizando um valor de chave que o dicionário não contém.

- **AttributeError (Erro de Atributo):** Você está tentando acessar um atributo ou método que não existe em um objeto.

- **IndexError (Erro de Índice):** O índice que você está usando para acessar uma lista, string ou tupla não existe no objeto, ou seja, é maior que seu comprimento menos um.

Imediatamente antes do ponto do erro, adicione uma delaração print para mostrar o valor do índice e o comprimento do objeto.

O objeto é do tamanho correto?

O índice está com o valor adequado?

E os erros de lógica ou de semântica?

Itremos utilizar o depurador ou debugger para observar os valores das variáveis conforme a execução de cada comando ocorre.

No PyCharm:
https://www.jetbrains.com/help/pycharm/2017.1/debugging.html

No IDLE:
http://ptcomputador.com/P/python-programming/94040.html

2. Testando o seu Código

- Quando escrevemos uma função, podemos escrever código para testá-la.

- Queremos saber se caso ela receba o que está previsto, fará o que é esperado.

- Utilizando este tipo de teste facilita a manutenção do seu código. Após alterações, os testes podem ser executados novamente.

- Utilizaremos o módulo **unittest** do Python.

2.1 Exemplo

- Vamos criar um caso de teste e verificar se um conjunto de entradas resulta na saída desejada.

- Para aprender a testar, precisamos de um código para testes. Eis uma função simples que aceita um primeiro nome e um sobrenome e devolve um nome completo formatado de modo elegante:

name_function.py

```
def get_formatted_name(first, last):
    """Gera um nome completo formatado de modo
elegante."""
    full_name = first + ' ' + last
    return full_name.title()
```

python™

2.1 Programa Principal

```
Enter 'q' at any time to quit.

Please give me a first name: janis
Please give me a last name: joplin
    Neatly formatted name: Janis Joplin.

Please give me a first name: bob
Please give me a last name: dylan
    Neatly formatted name: Bob Dylan.

Please give me a first name: q
```

names.py

```
from name_function import get_formatted_name

print("Enter 'q' at any time to quit.")
while True:
  first = input("\nPlease give me a first name: ")
  if first == 'q':
    break
  last = input("Please give me a last name: ")
  if last == 'q':
    break

  formatted_name = get_formatted_name(first, last)
  print("\tNeatly formatted name: " + formatted_name +
'.')
```

2.2 Testes de Unidade e Casos de Teste

- Um teste de unidade verifica se um aspecto específico do comportamento de uma função está correto.

- Um caso de teste é uma coleção de testes unitários que, em conjunto, prova que uma função se comporta como deveria em todas as situações esperadas.

- Um bom caso de teste considera todos os tipos possíveis de entradas que uma função pode receber e inclui testes que representem estas situações.

- Normalmente, se criam casos de teste para situações críticas no código do projeto.

2.3 Exemplo de Caso de Teste

```
import unittest
from name_function import get_formatted_name

 class NamesTestCase(unittest.TestCase):
  """Testes para 'name_function.py'."""

  def test_first_last_name(self):
    """Nomes como 'Janis Joplin' funcionam?"""
        formatted_name = get_formatted_name('janis',
'joplin')
         self.assertEqual(formatted_name, 'Janis
Joplin')

unittest.main()
```

python™

2.4 Métodos de Asserção

Tabela 11.1 – Métodos de asserção disponíveis no módulo `unittest`

Método	Uso
`assertEqual(a, b)`	Verifica se `a == b`
`assertNotEqual(a, b)`	Verifica se `a != b`
`assertTrue(x)`	Verifica se `x` é `True`
`assertFalse(x)`	Verifica se `x` é `False`
`assertIn(item, lista)`	Verifica se *`item`* está em *`lista`*
`assertNotIn(item, lista)`	Verifica se *`item`* não está em *`lista`*

python™

Tarefas

- 1 – Cidade, país: Escreva uma função que aceite dois parâmetros: o nome de uma cidade e o nome de um país. A função deve devolver uma única string no formado Cidade, País, por exemplo, Santiago, Chile. Armazene a função em um módulo chamado city_functions.py. Crie um arquivo de nome test_cities.py que teste a função que você acabou de escrever (lembre-se de que é necessário importar unittest e a função que você quer testar). Escreva um método chamado test_city_country() para conferir se a chamada à sua função com valores como 'santiago' e 'chile' resulta na string correta. Execute test_cities.py e garanta que test_city_country() passe no teste.

- 2 – População: Modifique sua função para que ela exija um terceiro parâmetro, population. Agora ela deve devolver uma única string no formato Cidade, País – população xxx, por exemplo, Santiago, Chile – população 5000000. Execute test_cities.py novamente. Certifique-se de que test_city_country() falhe dessa vez. Modifique a função para que o parâmetro population seja opcional. Execute test_cities.py novamente e garanta que test_city_country() passe novamente. Escreva um segundo teste chamado test_city_country_population() que verifique se você pode chamar sua função com os valores 'santiago', 'chile' e 'population=5000000'. Execute test_cities.py novamente e garanta que esse novo teste passe.