Programação em Python

Deteção e correção de erros

2023

Departamento de Ciência de Computadores

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Conteúdo

Erros

- Diferentes tipos de erros:
  - o programa é rejeitado pelo interpretador;
  - o programa dá uma mensagem de erro durante a execução;
  - não dá mensagens de erro, mas não faz o que é pretendido.
- São responsabilidade do programador: o computador tem sempre razão!
- Todos os programadores devem aprender a detetar e corrigir erros.
- É importante aprender a corrigir erros de forma construtiva.
- Não devem tentar fazer modificações ao acaso!

Tipos de erros

Sintático: o programa é rejeitado pelo interpretador Python

Execução: a execução aborta com uma mensagem de erro (exceção)

Semântico: o programa não dá o resultado que é pretendido

Erros sintáticos

Erros sintáticos

SyntaxError

```
  File "exemplo.py", line 15
    ....
SyntaxError: invalid syntax
```

- Indica a linha onde o erro é detetado
- Mas: a causa do erro pode estar nas linhas anteriores
- Alguns erros sintáticos comuns:
  - esquecer de fechar parêntesis ou aspas
  - usar uma palavra reservada como variável
  - esquecer dois-pontos (`:`) no início de um bloco
  - usar `=` em vez de `==`
  - usar indentação inconsistente

Corrigir erros sintáticos

- Observar atentamente a linha onde o erro é detetado
- Procurar parêntesis ou aspas abertos nas linhas anterioes
- Verificar a estrutura dos blocos if-elif-else, while e for
- Nos editores `IDLE` ou `Pyzo`: cores diferentes indicam palavras chave, cadeias de caracteres, etc.

Erros de execução

Erros de execução

Exceções: a execução aborta com uma mensagem de erro

Ciclo infinito: a execução não termina!

Erros de execução mais comuns

NameError

```
import math
def funcao(x):
    return math.sqrt(y)
```

```
>>> funcao(4)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/exemplo.py", line 3, in funcao
    return math.sqrt(y)
NameError: name 'y' is not defined
```

Erros de execução mais comuns (cont.)

TypeError

```
import math
def funcao(x):
    return math.sqrt(x)
```

```
>>> funcao("ola")
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/exemplo.py", line 3, in funcao
    return math.sqrt(x)
TypeError: a float is required
```

Erros de execução mais comuns (cont.)

ValueError

```
import math
def funcao(x):
    return math.sqrt(x)
```

```
>>> funcao(-1)
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/exemplo.py", line 3, in funcao
    return math.sqrt(x)
ValueError: math domain error
```

Erros de execução mais comuns (cont.)

IndexError

```
import math
def funcao(x):
    return x[0] + math.sqrt(x[1])
```

```
>>> funcao([10])
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "/exemplo.py", line 3, in funcao
        return x[0] + math.sqrt(x[1])
IndexError: list index out of range
```

Exemplo: verificar palíndromos

```
def palindromo(str):
  # Verifica se uma cadeia é igual ao seu reverso
  i = 0
  j = len(str)
  while i<j:
      if str[i]!=str[j]:
          return False
      i = i+1
      j = j-1
  return True
```

Erros de execução

```
>>> palindromo("olamundo")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/tmp/python-18422-ah.py", line 6,
in palindromo
IndexError: string index out of range
```

IndexError uso de índice inválido (de lista ou cadeia)

Traceback indica a instrução que causou a exceção

Corrigir erros de execução

```
def palindromo(str):
    ...
    print("i=", i, "j=", j, "len=", len(str))
    if str[i]!=str[j]:          # linha 6
    ...
```

1. Identificar o ponto onde erro foi detetado

Corrigir erros de execução

```
def palindromo(str):
    ...
    print("i=", i, "j=", j, "len=", len(str))
    if str[i]!=str[j]:          # linha 6
    ...
```

1. Identificar o ponto onde erro foi detetado
2. Acrescentar código para verificar valores de variáveis

Corrigir erros de execução (2)

Executando novamente:

```
>>> palindromo("olamundo")
i= 0 j= 8 len= 8
Traceback (most recent call last):
  ...
IndexError: string index out of range
```

Índices válidos:

$$0 \leq i, j \leq len - 1$$

Temos:

$$i = 0,\ j = 8,\ len = 8$$

Logo: o valor de j está errado!

Corrigir erros de execução (3)

```
def palindromo(str):
  # Verifica se uma cadeia é igual ao seu reverso
  i = 0
  j = len(str)  # erro: j>len(str)-1
  while i<j:
    if str[i]!=str[j]:
       return False
    i = i+1
    j = j-1
  return True
```

Corrigir erros de execução (4)

```
def palindromo(str):
  # Verifica se uma cadeia é igual ao seu reverso
  i = 0
  j = len(str)-1 # erro: j>len(str)-1 corrigido!
  while i<j:
    if str[i]!=str[j]:
       return False
    i = i+1
    j = j-1
  return True
```

Asserções

`assert` condição, mensagem

Declaração de uma condição que se deve verificar.

Se a condição for verdadeira passa a asserção e não faz mais nada

Se a condição for falsa aborta a execução com uma exceção `AssertionError: mensagem`

Acrescentar asserções

```
def palindromo(str):
  # Verifica se uma cadeia é igual ao seu reverso
  i = 0
  j = len(str)     # erro original
  while i<j:
    assert 0<=i<len(str), "limite do índice i"
    assert 0<=j<len(str), "limite do índice j"
    if str[i]!=str[j]:
        return False
    i = i+1
    j = j-1
  return True
```

Execução com asserções

AssertionError

```
>>> palindromo("olamundo")
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "/tmp/python-9298rhy.py",
line 8, in palindromo
AssertionError: limite do índice j
```

Quando a execução não termina

`RuntimeError`

- Problemas de programação elementares são resolvidos rapidamente num computador atual
- Se o programa demora muito tempo, provavelmente tem um ciclo infinito
- Em Python: recursão não limitada termina com uma exceção

```
RuntimeError: maximum recursion depth exceeded
```

Evitar ciclos infinitos

```
i = 0
while i<len(xs):
    # corpo do ciclo
    # deve modificar i e/ou xs
    ...
    # verifica os valores na condição
    print("i=", i, "len(xs)=", len(xs))
```

- O ciclo só termina quando a condição for `False`

Evitar ciclos infinitos

```
i = 0
while i<len(xs):
    # corpo do ciclo
    # deve modificar i e/ou xs
    ...
    # verifica os valores na condição
    print("i=", i, "len(xs)=", len(xs))
```

- O ciclo só termina quando a condição for `False`
- Acrescentamos código no fim do ciclo para verificar as actualizações das variáveis

Evitar ciclos infinitos (2)

- Verificar se as variáveis na condição `while` são alteradas no corpo do ciclo
- Pensar quais os intervalos desejados para índices
- Se possível: escrever um ciclo simples em vez de dois ciclos embricados
- Se possível: usar um ciclo `for` (termina sempre)

Erros semânticos

Erros semânticos

- O programa dá um resultado, mas não é o esperado
- Logo: o programa que escrevi não resolve o problema pretendido
- O erro pode estar:
  - na compreensão do problema
  - no modelo mental de como construir a solução (algoritmo)
  - na tradução do algoritmo para Python

Corrigir erros semânticos

- Verifique se compreendeu bem o problema
- Pensar antecipadamente em casos de teste simples: valores para os dados e para o resultado
- Dividir para conquistar: decomponha o problema em funções
- Cada função deve ter um objetivo bem definido
- Efetue testes das funções no interpretador de Python

Testes unitários

- Especificação do valor esperado de funções/métodos/etc. para valores concretos de argumentos
- Servem de documentação do programa
- Podem ser testados automaticamente
- Módulo *doctest*:
  - testes incluídos na documentação das funções
  - sintaxe pergunta/resposta semelhante ao interpretador de Python
  - usado no sistema *Codex* de submissões automáticas

Exemplo: factorial

```
def factorial(n):
    '''
    Calcula o factorial de n.
    Exemplos:
    >>> factorial(0)
    1
    >>> factorial(1)
    1
    >>> factorial(2)
    2
    >>> factorial(3)
    6
    '''
    p = 1
    for i in range(1,n):
        p = p * i
    return p
```

Testar um módulo

```
>>> import doctest
>>> doctest.testmod()
```

Não produz mensagens se os testes passam.

Quando um teste falha

```
>>> import doctest
>>> doctest.testmod()
**********************************************************************
File "__main__", line 8, in __main__.factorial
Failed example:
    factorial(2)
Expected:
    2
Got:
    1
⋮

1 items had failures:
   2 of   4 in __main__.fatorial
***Test Failed*** 2 failures.
```

Alguns erros comuns

Cuidado com as comparações com três operandos.

```
# programa original
if not (a==b==c):
    ⋮
```

```
# errado!
if a!=b!=c:
    ⋮
```

```
# correto
if a!=b or b!=c:
    ⋮
```

É preferivel usar comparações simples: `a==b and b==c` em vez de `a==b==c`.

Alguns erros comuns

Os limites dum ciclo `for` não devem ser re-calculados dentro do ciclo.

```
for i in range(len(xs)): # remover valores pares
    if xs[i]%2==0:
        del xs[i]  # erro
```

Alguns erros comuns

A palavra reservada `else` também pode ser associada a um ciclo (não usamos nestas aulas).

```
# while/else
while ...:
    if ...:
        ...
else:
    ...
```

```
# if/else
while ...:
    if ... :
        ...
    else:
        ...
```

Alguns erros comuns

Duplicar o nome de uma lista em vez do seu conteúdo.

```
>>> xs = [1,2,3,4,5]
>>> ys = xs
>>> del xs[0]
>>> ys
[2,3,4,5]
```

```
>>> xs = [1,2,3,4,5]
>>> ys = xs[:]
>>> del xs[0]
>>> ys
[1,2,3,4,5]
```