Programação I / Introdução à Programação

Capítulo E, "Exceptions"

João Pedro Pedroso

2024/2025

Últimas aulas

- Programação orientada a objetos
  - `class`
    - instância/objeto: variável de uma determinada classe
    - método: função definida dentro de uma classe
    - atributo: um dos dados que constituí uma instância
  - métodos *"mágicos"*:
    - método de inicialização/construtor: `__init__(self, ...)`
    - conversão para *string*: `__str__(self)`:
    - operadores:

Método	Uso	Descrição
`__add__`	`x + y`	soma
...		
`__lt__`	`x < y`	`True` se x for menor do que y
`__eq__`	`x == y`	`True` se x for igual a y
...		

Hoje:

- Exceções

Exceções

- Sempre que o Python encontra um erro de execução, cria um objeto de exceção
- A execução do programa é interrompida, Python imprime o *traceback* (linhas de execução anteriores) e uma mensagem a descrever a exceção que ocorreu
- Última linha:
  nome do erro: detalhes acerca do erro

```
>>> f = open("secretbox.txt", "r")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'secretbox.txt'
```

Exceções: exemplos:

- divisão por zero:

```
>>> print(55/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

- acesso a um índice que não existe numa lista:

```
>>> a = []
>>> print(a[5])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

- atribuir um valor a um elemento de um tuplo:

```
>>> tup = ("a", "b", "d", "d")
>>> tup[2] = "c"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tratamento de exceções

- Por vezes é necessário escrever código que pode dar origem a erros
- Para que o programa não termine, em situação de erro, é necessário tratar a exceção
- Em Python, na forma mais simples:

```
try:
    # code that may raise an error
    ...
except ExceptionName:
    # code to execute in case exception "ExceptionName" arises
```

- Exemplos de exceções pré-definidas:

ValueError	argumento fora do domínio; e.g., `sqrt(-1)`
IndexError	índice fora de limites
KeyError	chave de dicionário inexistente
TypeError	erro de tipos, e.g. `"a"+3`
FileNotFoundError	erro a abrir ficheiro para leitura
IOError	erro na leitura/escrita de dados

Exemplo: erro a abrir ficheiro

```
filename = input("Enter a file name: ")
try:
    f = open(filename, "r")
except FileNotFoundError:
    print("There is no file named", filename)
    ...
    # what to do: try another name, exit, ...?
```

Instrução try: forma completa

```
user_input = input('Type a number: ')
try:
    # Try to do something that could fail.
    user_input_as_number = float(user_input)
except ValueError:
    # This will be executed if a ``ValueError`` is raised.
    print('You did not enter a number.')
else:
    # This will be executed if no exception got
    # raised in the ``try`` statement.
    print('The square of your number is', user_input_as_number**2)
finally:
    # This will be executed whether or not an exception is raised.
    print('Thank you')
```

- Output sem erro:

```
Type a number: 27
The square of your number is 729.0
Thank you
```

- Output com erro:

```
Type a number: vinte e sete
You did not enter a number.
Thank you
```

Detalhe da execução da instrução composta `try`

- primeiro é executada a cláusula `try`
  - i.e., o bloco entre `try` e `except`
- se não ocorrerem exceções na execução da cláusula `try`:
  - o Python ignora os blocos `except ...`
  - se existir, o bloco `else` é executado
- se ocorrer uma exceção:
  - as linhas do bloco `try` a seguir àquela onde apareceu a exceção são ignoradas
  - procura-se um bloco `except ...` correspondente à exceção
    - se existir, é executado → a exceção é tratada
    - se não existir → a exceção não é tratada
- em ambas as situações:
  - se existir, o bloco `finally` é executado
    - se a exceção não foi tratada, ocorre o erro correspondente
- se a exceção foi tratada, o programa continua na linha a seguir à instrução `try`
  - caso contrário, termina com o erro correspondente à exceção

`try`: Notas de utilização

- Os blocos a seguir a `except` são opcionais (`else`, `finally`)
- O bloco `try` deve ser tão pequeno quanto possível
  - caso contrário, poderá haver muitas causas diferentes (e imprevistas) para exceções
- Se o bloco `try` puder falhar de várias formas, pode-se tratar várias exceções na mesma instrução
  - vários blocos `except`
- Se não se especificar a exceção (linha `except:` como abaixo)
  - bloco trata qualquer exceção

```
try:
    ...
except:   # any exception fits here
    ...
```

  - a evitar... pode facilmente esconder erros

```
try:
    txt = input("enter something ")
except EOFError:
    print("end of file during input")
except KeyboardInterrupt:
    print("interrupted...")
except:
    print("something weird occurred")
else:
    print("you entered", txt)
finally:
    print("thanks")
```

Cláusula `finally` da instrução `try`

- Padrão frequente em programação:
  1. reservar um recurso, eg:
     - janela onde se desenha tartarugas
     - ligação internet
  2. executar série de instruções que poderão ou lançar exceções
  3. quer haja exceções ou não: libertar recursos reservados
     - fechar janela
     - desligar internet
- normalmente, o programa irá prosseguir na linha a seguir à instrução `try`:
- problema: caso persista um erro, e o programa deve terminar
- cláusula `finally` permite libertar recursos antes de terminar

Lançar exceções

Lançar exceções

- Podemos lançar exceções, para interromper o programa em situações que o impeçam de prosseguir corretamente
- Instrução `raise`, eg:

```
raise ValueError("cannot determine factorial of a negative number")
```

- Lista de todas a exceções predefinidas em Python: `https://docs.python.org/3/library/exceptions.html`

Exemplo

```python
def factorial(n):
    if n<0:
        raise ValueError("cannot determine factorial of a negative number")
    if type(n) != int:
        raise ValueError(f"'factorial' cannot be called with value {n}")
    p=1
    while n>0:
        p = p*n
        n = n-1
    return p
```

```
>>> factorial(-2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in factorial
ValueError: cannot determine factorial of a negative number
>>> factorial(1.5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 5, in factorial
ValueError: 'factorial' cannot be called with value 1.5
>>> factorial(10)
3628800
```

Noções estudadas

exception An error that occurs at runtime.

handle an exception To prevent an exception from causing our program to crash, by wrapping the block of code in a `try... except` construct.

raise To create a deliberate exception by using the raise statement.

Aula de hoje:

- Exceções

Próximas aulas

- Módulos do Python para cálculo científico: `NumPy`