Python 3

9. Errores y Excepciones

Carolina Mañoso, Ángel P. de Madrid y Miguel Romero

Índice

- Errores
- Tratamiento de Excepciones
- Generación de Excepciones

Errores

- Los errores de sintaxis son *errores que lanza el intérprete cuando no reconoce el código escrito*.
  - El intérprete muestra donde se produce el error y una descripción del mismo.

```
>>> print("hola)
SyntaxError: EOL while scanning string literal

>>> while True print("hola")
SyntaxError: invalid syntax
```

Tratamiento de Excepciones (1/8)

- Las excepciones son *errores detectados en tiempo de ejecución*.
  - El programa se detiene temporalmente y se genera un dato de tipo `Exception`.
  - Esta excepción se puede capturar y tratar, para después continuar con el programa.
  - Si no se captura, se acaba el programa y se genera un mensaje de error por consola.

Tratamiento de Excepciones (2/8)

- El mensaje de error indica el tipo de excepción y una explicación.

```
>>>5/0
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    5/0
ZeroDivisionError: division by zero
```

```
>>>"a"+2
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    "a"+2
TypeError: Can't convert 'int' object to str implicitly
```

```
>>>3+x
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    3+x
NameError: name 'x' is not defined
```

Tratamiento de Excepciones (3/8)

- Si en una parte del código puede haber una excepción, se debe programar qué hacer en el caso de que ocurra. `try-except` captura y trata las excepciones:
  - `try`: se define el fragmento de código en el que creemos que podría producirse la excepción.
  - `except`: permite indicar el tratamiento que se llevará a cabo de producirse una excepción.

```
try:
    codigo donde podría haber un problema
except type_of_error:
    codigo a ejecutar si hay un error

try:
    l = a/b
except ZeroDivisionError:
    print("Cambia el denominador")
```

Tratamiento de Excepciones (4/8)

- Para encontrar el tipo de error que puede tener lugar, se provoca la excepción:

```
Traceback (most recent call last):
    File "<pyshell#0>", line 1, in <module>
       num = int(input("Enter a number: "))
ValueError: invalid literal for int() with base 10:'hola'
```

Tratamiento de Excepciones (5/8)

- La lista de las excepciones es de forma jerárquica:

https://docs.python.org/3.8/library/exceptions.html

- La jerarquía implica el nivel de generalidad de la excepción:
  - Cuanto más arriba en el árbol, más generales.
  - Cuanto más abajo, más específicas.

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
```

Tratamiento de Excepciones (6/8)

- Una declaración `try` puede tener mas de un `except` para tener manejadores para distintas excepciones. Sólo uno será ejecutado (busca en orden).
  - Se deben colocar las excepciones *de lo más específico a lo más general*.

```
try:
    y = 1/0
except ZeroDivisionError:
    print("Es Division por Cero")
except ArithmeticError:
    print("Error Aritmético")
```

Nota: Pruebe a ejecutarlos en el otro orden.

Tratamiento de Excepciones (7/8)

- El último `except` puede omitir el nombrar el tipo de excepción, para servir de comodín.
- Esta construcción puede tener un bloque `else` opcional después de los `except` que se ejecutará si no se genera excepción.

```
try:
    x = int(input("Introduzca un número: "))
except ValueError:
    print("Eso no es un número")
except:
    print("Error inesperado")
else:
    print("Muy bien")
```

Tratamiento de Excepciones (8/8)

- `try` tiene la cláusula opcional `finally` que se ejecuta siempre antes de salir de la declaración, con la intención de realizar tareas de *limpieza*.

```
def dividir(x,y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("División por cero!!!")
    else:
        print("El resultado es", result)
    finally:
         print("Ejecuto finally")

dividir(6,3)
dividir(6,0)
```

Tratamiento de Excepciones: Resumen

try: Contiene el código que puede producir la excepción.

except TipoError: Contiene el código que gestionará un tipo de error concreto.

except: Contiene el código que gestionará cualquier error.

else: Contiene el código que se ejecutará en caso de no haber error.

finally: Contiene el código que se ejecutará en cualquier caso.

Práctica: Tratamiento de Excepciones

- Calcule el factorial de un número introducido por teclado.
  - Gestione las excepciones.
    - Excepción de interrupción: `KeyboardInterrupt`.
    - Excepción general.
    - Cláusula `else`.
    - Cláusula `finally`.

Nota: Recuerde que el factorial de un número entero n es $n \times (n - 1) \times (n - 2) \times \ldots \times 3 \times 2 \times 1$.

Práctica: Solución

```
#Cálcula el Factorial
try:
    numero = int(input('Introducir un número: '))
except KeyboardInterrupt:  # Captura excepción de interrupción
    print('\nSe ha pulsado ctrl+c') # Interrupción Ctrl+c
except:
    print('Debe introducir un número entero')
else:  # Se ejecuta si no hay error
    factorial = 1
    for num in range(1, numero+1):
        factorial *= num
    print(factorial)
finally:  # Se ejecuta tanto si hay error como si no
    print('Fin de programa')  # Muestra mensaje final
```

Generación de Excepciones (1/4)

- La declaración `raise` permite al programador forzar a que ocurra una excepción específica.

```
>>>raise NameError(" Hola ")
    Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    raise NameError(" Hola ")
NameError:  Hola
```

  - Si queremos que se ejecute algo aunque lancemos la excepción, introducimos dentro de la declaración `try` la cláusula `finally`.

```
>>>try:
       raise KeyboardInterrupt
   finally:
        print ("Adiós")

adios
  Traceback (most recent call last):
    File "<pyshell#6>", line 2, in <module>
      raise KeyboardInterrupt
  KeyboardInterrupt
```

Generación de Excepciones (2/4)

- El usuario puede definir sus propias excepciones.
- Deben derivar de la clase `Exception`.

exception **`Exception`**

All built-in, non-system-exiting exceptions are derived from this class. All user-defined exceptions should also be derived from this class.

- Hay que definir tantas clases como tipos de excepciones.
- Se suele crear una clase `base` que sea superclase para las clases con las excepciones específicas.

Generación de Excepciones (3/4)

```
class Error(Exception):
    """Clase base para tratar las excepciones"""
    pass


class TipoNoValido(Error):
    """Excepción lanzada cuando se introduce un tipo no
compatible con la clase vector"""

    #mensaje: explicación del error
    def __init___(self,mensaje):
        self.mensaje = mensaje
…
raise TipoNoValido("Parámetro de entrada no válido")
```

Generación de Excepciones (4/4)

- `Assert expresion` evalúa una expresión y, si es cierta, o no nula o… no sucede nada; en caso contrario, lanza una excepción llamada `AssertionError`.

```
import math
x = float(input())
assert x >= 0.0
x = math.sqrt(x)
print(x)
```

  - Se utiliza cuando queremos estar seguros de que no tenemos datos erróneos.
  - Al generar una excepción `AssertionError`, su código evita la aparición de resultados no válidos y muestra claramente la naturaleza del error.

Práctica: `Assert`

- Defina una lista con cinco elementos introducidos por teclado.
- A continuación, mediante un bucle se irán eliminando elementos, uno a uno, desde el final de la lista.
- La condición establecida para que no se produzca la excepción es que la lista tenga al menos un elemento. Cuando haya sólo un elemento y se intente borrar de nuevo se producirá la excepción **AssertionError**.

Práctica `Assert` : Solución

```
lista = [0,0,0,0,0]

for i in range(5):
    lista[i]= input('introduce item {}: '.format(i))
print("La lista introducida es:", lista)

try:
    while True:  # Bucle infinito hasta error
        print('Voy a borrar el elemento: ',lista[-1])
        lista.pop()  # Borra elemento
        assert len(lista) > 0
        print("La lista despues de borrar es:", lista)  #
Muestra lista después de borrado

except AssertionError:  # Excepción para assert
    print('Error al intentar borrar el ultimo elemento')
    print('La lista debe contener al menos 1 elemento')
```

Aviso

Esta colección de transparencias se distribuye con fines meramente docentes.

Todas las marcas comerciales y nombres propios de sistemas operativos, programas, hardware, etc. que aparecen en el texto son marcas registradas propiedad de sus respectivas compañías u organizaciones.