Python

Guía para ser un Pythonista

Juan José Lozano Gómez – j2logo.com

17

Manejo de errores y excepciones

Índice de contenidos

Introducción

Todos los programas son susceptibles a errores. Es algo que debes tener en cuenta. Por muy bien programada que esté una aplicación, siempre puede ocurrir un error ajeno: que el pc se quede sin memoria, que se haya borrado un fichero que se esperaba, que no haya conexión a Internet, ...

Pero no nos engañemos, antes o después, cualquier programador introduce algún pequeño error en el código sin darse cuenta, así que, hay que lidiar con ello.

En este tema veremos qué es una excepción, qué tipo de errores se pueden producir en una aplicación y cómo se pueden manejar o controlar los errores en Python.

Excepciones

Como te decía en la introducción, en cualquier programa se puede producir un error en el momento más inesperado.

En Python, cuando un error no se controla en una aplicación, esta termina inmediatamente.

Generalmente los errores se pueden clasificar en dos grupos:

- **Errores de sintaxis**. Se producen cuando el programador escribe mal una instrucción o una expresión.
- **Excepciones**. Son errores que se detectan o se producen durante la ejecución de un programa.

En el siguiente ejemplo se muestra un error de sintaxis:

```
>>> mayor_0 = 3
>>> while mayor_0
...     print(mayor_0)
...     mayor_0 -= 1
...
  File "<input>", line 1
    while mayor_0
                ^
SyntaxError: invalid syntax
```

En cuanto a las excepciones, estas se producen en tiempo de ejecución, es decir, una vez que el programa está en funcionamiento.

Como su propio nombre indica, una excepción se genera ante una situación inesperada en la cuál el programa no puede continuar su flujo normal. Sin embargo, esto no quiere decir que se produzca un error fatal y el programa tenga que terminar. En ocasiones, las excepciones son controladas (concepto conocido como manejo de excepciones), el error se puede corregir y, en ese caso, el programa continúa. En caso contrario, se muestra un mensaje con el fallo y el programa finaliza su ejecución.

Ejemplos de este tipo de excepciones se producen, por ejemplo, cuando se intenta dividir un número por cero, cuando se intenta acceder a un elemento en un objeto de tipo secuencia con un índice fuera del rango permitido o cuando se intenta realizar una operación con objetos de distinto tipo.

Observa el código siguiente:

```
>>> a = 3
>>> num = int(input())
>>> print(a/num)
```

Si el usuario introdujera el número `0`, se intentaría realizar la división `3/0`. Como sabrás, esto es imposible en programación. ¿Qué ocurre entonces? Básicamente, el programa mostraría el siguiente mensaje de error y finalizaría su ejecución.

```
Traceback (most recent call last):
  File
"/Users/Juanjo/workspace/proyectos/ejemplos_pyt
hon/exceptions.py", line 11, in <module>
    print(a/num)
ZeroDivisionError: division by zero
```

La última línea muestra la excepción que se ha producido (`ZeroDivisionError`) y un mensaje descriptivo (`division by zero`). Esto suele ser así para todas las excepciones que incorpora el lenguaje. Lo que precede a la última línea es la traza del error, en la cuál se puede identificar dónde se produjo exactamente.

`ZeroDivisionError` es una excepción que incorpora el lenguaje, pero no es la única. Python define una serie de excepciones incorporadas que están definidas en un módulo específico. Veremos algunas de ellas en la siguiente sección.

Excepciones incorporadas

Hay muchas excepciones incorporadas en Python que se generan cuando se produce un "error común".

Estas excepciones están definidas en el módulo `builtins`.

Puedes verlas todas si ejecutas el código siguiente:

```
>>> import builtins
>>> dir(builtins)
```

Algunas de estas excepciones son:

- `ImportError`: Se genera cuando no se encuentra el módulo importado.
- `IndexError`: Se genera cuando el índice de una secuencia está fuera de rango.
- `KeyError`: Se genera cuando no se encuentra una clave en un diccionario.
- `ZeroDivisionError`: Se genera cuando el segundo operando de la división u operación de módulo es cero.

Esos son solo algunos ejemplos. Hay muchas más.

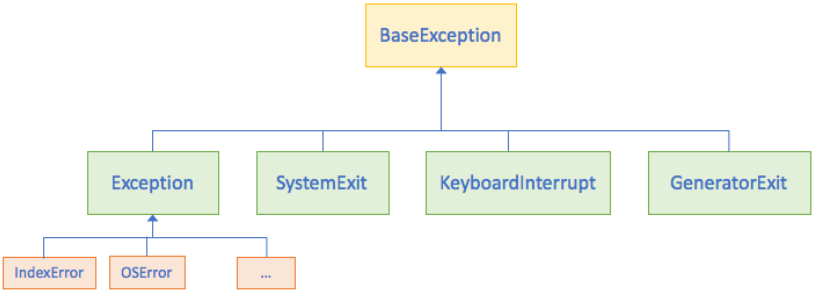
Normalmente, estas excepciones son lanzadas por el intérprete o por las funciones incorporadas de Python y suelen ir acompañadas de un mensaje que indica la causa del error.

No obstante, el código de usuario (el que tú programas) también puede hacer uso o lanzar este tipo de excepciones, aunque es probable que no se haga de la forma apropiada. Si quieres lanzar excepciones, lo mejor es que crees las tuyas propias, como veremos en una sección posterior.

Una característica fundamental que tienen todas las excepciones incorporadas es que heredan de la excepción base `BaseException`.

Sin embargo, se recomienda que las excepciones personalizadas hereden de la clase hija `Exception` (la mayoría de las excepciones incorporadas también heredan de esta).

En el siguiente diagrama puedes ver la jerarquía de clases de las excepciones incorporadas:



Manejo de excepciones

Bien, hemos hablado de excepciones, de situaciones inesperadas, que las excepciones se generan, que las excepciones pueden ser tratadas, que una excepción no controlada hace que un programa termine, ... Pongamos todo en orden.

Cuando en un programa se detecta una situación inesperada o una situación que impide el flujo de ejecución normal del mismo, en ese punto se crea una instancia (un objeto) de una excepción concreta y ***se lanza***. Por ejemplo, es lo que hace Python cuando detecta que un número se pretende dividir por cero.

¿Qué significa que *se lanza* una excepción? Básicamente que el intérprete de Python detiene el flujo de ejecución en el proceso actual y lo pasa al proceso invocador. Así

sucesivamente hasta que algún proceso capture y maneje la excepción. Si no se maneja, el programa finalizará.

Por ejemplo, considera un programa en el que hay una función *A* que llama a una función *B,* que a su vez llama a una función *C*. Si ocurre una excepción en la función *C,* pero no se maneja en *C*, la excepción pasa a *B*. Si en B no se maneja, entonces pasa a *A*.

Si nunca se maneja, se muestra un mensaje de error y el programa finaliza de forma inesperada.

El manejo de errores y excepciones generalmente se resuelve guardando el estado de ejecución en el momento en que ocurrió el error e interrumpiendo el flujo normal del programa para ejecutar un código conocido como *manejador de la excepción*. Dependiendo del tipo de error (*división por cero*, *error de apertura de archivo*, etc.) que se haya producido, el manejador de la excepción puede "solucionar" el problema y el programa continúa con los datos previamente guardados.

Manejo de excepciones en Python

En Python, las excepciones se pueden manejar usando la sentencia `try`.

El código que puede generar una excepción se coloca dentro de la sentencia `try`. El código que maneja las excepciones se escribe en la cláusula `except`.

De este modo, podemos elegir qué acciones realizar una vez que se haya detectado o lanzado una excepción.

Veamos el siguiente ejemplo:

```
import sys

valores = ['a', 0, 2]

for v in valores:
    try:
        print('El valor es', v)
        r = 1/int(v)
    except:
        print('Ocurrió un error',
              sys.exc_info()[0])
        print('Siguiente')
        print()
```

Como puedes observar, la expresión `1/int(v)` puede generar dos excepciones diferentes: una cuando `v` es `0` y otra cuando `v` no es un número. Por esta razón este código se ha colocado dentro de la sentencia `try`.

El manejo de la excepción, el bloque de instrucciones dentro de la sentencia `except`, simplemente muestra por pantalla la excepción que se ha producido (referenciada por `sys.exc_info()[0]`).

Si no se manejaran las excepciones, es decir, si no se hubiera incluido el bloque `try… except…`, en la primera iteración del bucle, cuando `v` hace referencia al string `'a'`, el programa finalizaría inesperadamente. Sin embargo, al haber manejado las excepciones, la salida del programa es esta:

```
El valor es a
Ocurrió un error <class 'ValueError'>
Siguiente

El valor es 0
Ocurrió un error <class 'ZeroDivisionError'>
Siguiente

El valor es 2
0.5
```

Dado que todas las excepciones heredan de la excepción base `Exception`, el código siguiente es equivalente al código que te he mostrado anteriormente:

```
valores = ['a', 0, 2]

for v in valores:
    try:
        print('El valor es', v)
        print(1/int(v))
    except Exception as e:
        print('Ocurrió un error',
              e.__class__)
        print('Siguiente')
        print()
```

Como ves, se puede tener acceso a la instancia de la excepción añadiendo `as` y el nombre de una variable después del nombre de la excepción capturada.

Cómo funciona la sentencia try... except...

La sentencia `try` funciona de la siguiente manera.

- En primer lugar, se ejecutan las instrucciones entre las sentencias `try ... except ...`

- Si no se produce ninguna excepción, se omite la cláusula `except` y finaliza la ejecución de la instrucción `try`.

- Si se produce una excepción durante la ejecución de la cláusula `try`, se omite el resto de la cláusula.

    - Si la excepción es de una clase (o una subclase) que coincide con alguna de las excepciones indicadas por la cláusula `except`, se ejecuta la cláusula `except` y, a continuación, la ejecución del programa continúa justo tras la instrucción `try`.

    - En caso contrario, la excepción se va propagando por los bloques de código invocadores para que pueda ser manejada por alguno de ellos. Si finalmente no se encuentra ningún manejador, se trata de una excepción no controlada y la ejecución del programa finaliza con un mensaje como se muestra arriba.

Indicando más de una cláusula except

Una instrucción `try` puede definir más de una cláusula `except`, de manera que es posible especificar distintos

manejadores para las diferentes excepciones que se puedan dar. Como máximo se ejecutará un manejador por cada sentencia `try` que aparezca.

A continuación, te muestro un ejemplo de sentencia try que maneja varias excepciones:

```
try:
    # Código que puede
    #lanzar una excepción

except ValueError:
    # Maneja la excepción ValueError

except (TypeError, ZeroDivisionError):
    # Maneja las excepciones
    # TypeError y ZeroDivisionError

except:
    # Maneja el resto de excepciones
```

IMPORTANTE: Es una buena práctica de programación especificar el tipo de excepción que se maneja. Cuidado con definir una cláusula `except` sin indicar la excepción o su forma equivalente `except Exception`. En este caso puedes ocultar errores de programación reales, además de que tratarías todos los errores de la misma manera.

Lanzar una excepción

Ya hemos visto que, cuando Python detecta una situación inesperada o anómala este lanza una excepción.

No obstante, tú, como programador, también tienes la capacidad de lanzar excepciones cuando en tu código detectes un caso que impida la ejecución "normal" del programa. Para ello se utiliza la sentencia `raise` seguida del nombre de la excepción.

Imagina un script en el que un usuario debe introducir un número entero positivo. Dicho script podría ser como sigue:

```
try:
    n = int(input('Introduce un número > '))
    if n < 0:
        raise ValueError('El número es
                          negativo')
    print('El número introducido es', n)
except ValueError as ve:
    print(ve)
    raise
```

Como puedes apreciar, cuando el usuario introduce un número menor que cero, el programa lanza la excepción `ValueError`. Al lanzar una excepción, es posible pasar como argumento un mensaje indicando el motivo del error.

Fíjate también en que en el manejador de la excepción se ha añadido la sentencia `raise` sin indicar nada más. Cuando esto ocurre en un manejador, la misma excepción continúa propagándose hacia niveles de código superior.

Cláusula else

En algunas situaciones, es posible que tengas que ejecutar un determinado bloque de código solo si las instrucciones dentro del `try` no lanzaron ningún error.

Para estos casos, se usa la cláusula opcional `else` junto con la instrucción `try`.

Nota: Las excepciones que se puedan lanzar en la cláusula `else` no son manejadas por las cláusulas `except` previas.

```
try:
    n = int(input('Introduce un número > '))
    if n == 0:
        raise ValueError('Valor no válido')
except ValueError as ve:
    print(ve)
else:
    print(5/n)
```

Cláusula finally: liberando recursos

La instrucción `try` en Python también puede incluir la cláusula opcional `finally`. Esta cláusula se ejecuta siempre, independientemente de que se haya producido o no una excepción y de que se haya

manejado o no. Generalmente se usa para liberar recursos externos.

Por ejemplo, imagina que estas conectado a un centro de datos remoto a través de la red o trabajas con un archivo o una interfaz gráfica de usuario (GUI).

En todas estas circunstancias, debes liberar los recursos antes de que el programa se detenga, ya sea que se haya ejecutado con éxito o no. Estas acciones (cerrar un archivo, GUI o desconectarse de la red) se realizan en la cláusula `finally` para garantizar la ejecución.

Sé que todavía no hemos visto el manejo de archivos (lo haremos en el tema siguiente), pero aquí te muestro un ejemplo sencillo de lo que te acabo de comentar:

```
try:
    f = open("test.txt", encoding='utf-8')
    # Operaciones
finally:
    f.close()
```

Este tipo de construcción asegura que el archivo se cierre, incluso si se produce una excepción durante la ejecución del programa.

Excepciones personalizadas

Ya te he comentado que una buena práctica cuando estás desarrollando una aplicación es usar tus propias excepciones en lugar de las que Python incorpora.

Para ello, simplemente debes implementar una clase que herede directa o indirectamente de la clase `Exception`.

Normalmente, estas clases deben ser lo más simples posible. ¿Recuerdas cuando vimos la sentencia `pass`? Pues es muy usada a la hora de crear excepciones.

A continuación, te muestro un ejemplo de varias excepciones personalizadas definidas para una aplicación:

```
class AppBaseError(Exception):
    pass

class ObjectNotFoundError(AppBaseError):
    pass

class InvalidDateError(AppBaseError):
    pass
```

Buenas prácticas

- Agrupa las excepciones de la aplicación en un módulo llamado `exceptions.py` o `errors.py`.
- Si la aplicación es muy grande y contiene muchos paquetes, puedes agrupar las excepciones relacionadas en diferentes módulos a través de los distintos paquetes.
- Implementa una excepción base personalizada que herede de `Exception`.
- Acaba los nombres de las excepciones con la palabra `Error`.

En los videotutoriales del tema puedes ver más ejemplos de excepciones personalizadas y cómo pasar información adicional del error.