Tema 7: Control de errores

Apuntes

[**Errores y excepciones 2**](#_b9n01yoiydmc)

[**Concepto de Excepción en Python 2**](#_s8fykzq4gjqs)

[**Recoger y manejar excepciones 3**](#_minwzji3yzq9)

[**Excepciones predefinidas 4**](#_x4i6eupmotno)

[**Disparar/lanzar y propagación 5**](#_x6zq4pdngwcd)

[**Jerarquía de excepciones 5**](#_pe9mqv69usok)

[**Excepciones de usuario 5**](#_dfncekx20nj9)

# Errores y excepciones

Tipos de errores:

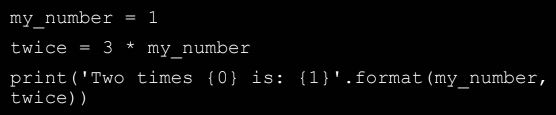
* Syntax error



* Semantics



* Error of concept or requirements



Se pueden detectar los errores antes de la ejecución, en el momento de la ejecución o mediante test.

# Concepto de Excepción en Python

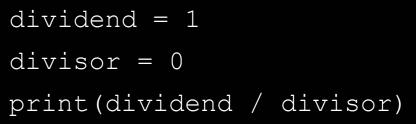
Nombre dado a un tipo de error específico que ocurre durante la ejecución de un programa.

Los errores semánticos suelen llevar a excepciones.

En python se llaman errores.

Ejemplo:

Consecuencia: error fatal y aborta el programa.



Tipos:

* Aritmético
* Fuera de límites
* Tipo incorrecto
* …

En python los nombres de las excepciones se nombran normalmente con error al final (ArithmeticError, ValueError...). Hay excepciones a esta regla como Exception, BaseException, KeyboardInterrupt…

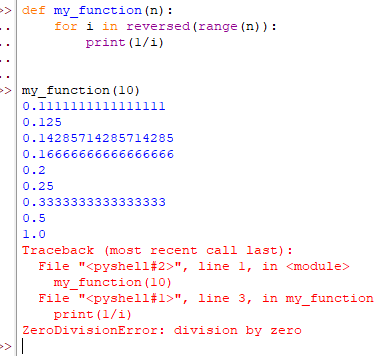
Cuando ocurre un error/excepción se da cierta información:

* Donde ha ocurrido (linea y punto en el script)
* Lo que lo ha causado
* Traza del error (en donde se encuentra dentro de una estructura anidada)

Ejemplo:

Dará error al llegar a 0.

Se puede ver que en la traza indica la función y la línea en la que se produce el error.



# Recoger y manejar excepciones

Para atrapar un error o excepción se usará una “trampa” con:

**try → except** → else → finally

Gracias a esto un programa puede seguir ejecutandose si hay un error pero dentro del try

**Sintaxis:**

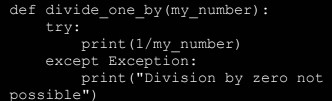
try:

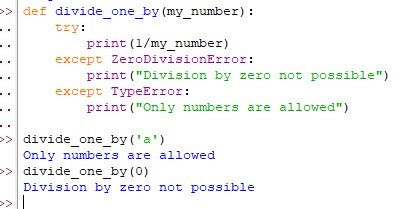
<código “problemático”>

except <Excepción esperada>:

<código si se produce el error>

Ejemplo de ejecución:

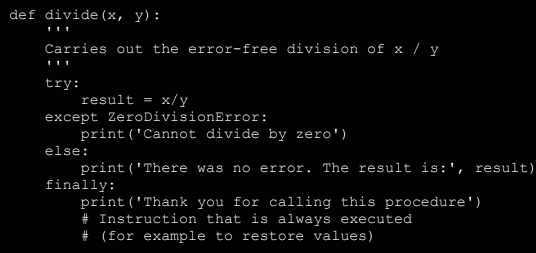




De las dos opciones es preferible la segunda, con más excepciones dedicadas a diferentes errores.

**Ejemplo que muestra todas las partes del try:**

Procedimiento que imprime por pantalla la división de dos números, se prueba a hacer la división (**try**) y se contemplan errores, declarando una excepción, en el **else** se devuelve lo que saldría si no hay error (opcional) y el **finally** se ejecuta siempre, haya error o no



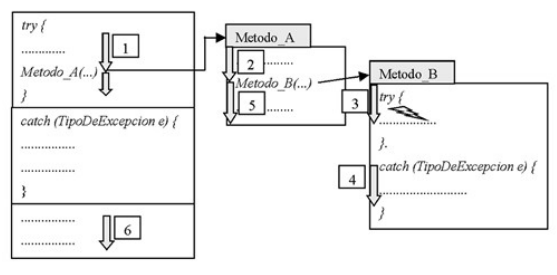
# Excepciones predefinidas

Son con las que cuenta el lenguaje para describir sus errores.

Documentación:  
<https://docs.python.org/3/library/exceptions.html>

<https://www.w3schools.com/python/python_ref_exceptions.asp>

# Disparar/lanzar y propagación



# Jerarquía de excepciones

La jerarquía de excepciones se estructura en forma de árbol, cuanto más cerca están de la raíz/tronco más generales serán y capturarán más errores. Se debe intentar usar siempre cosas más específicas, es decir, más lejos de la raíz/tronco.



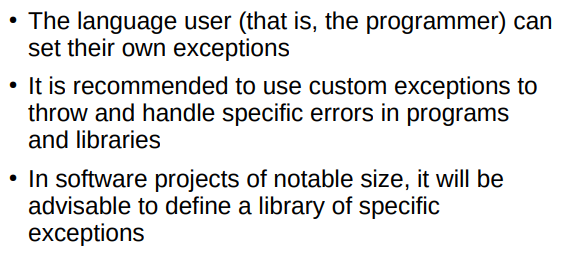
**Extract from the built-in exception hierarchy:**

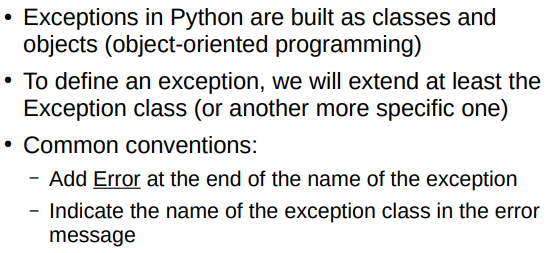
(buscar definición de cada)

* BaseException
  + Exception
    - KeyboardError
    - StopAsyncIteration
    - StopIteration
    - ImportError
      * ModuleNotFoundError
      * ZipImportError
      * ...
  + AssertionError
  + ArithmeticError
    - FloatingPointError
    - OverflowError
    - ZeroDivisionError
  + KeyboardInterrupt

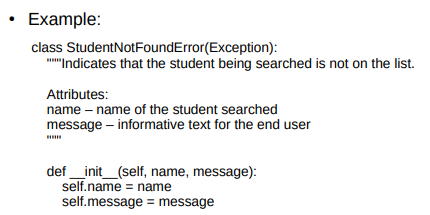
Jerarquía completa: <https://docs.python.org/3/library/exceptions.html#exception-hierarchy>

# Excepciones de usuario





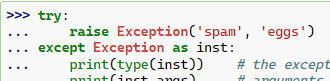
**Crear una excepción:**



Parámetro: Exception → será en la jerarquía de excepciones la excepción madre

El contenido será un método, no una función. El nombre será \_\_init\_\_ y los parámetros siempre serán self y los parámetros que necesitemos. El contenido puede variar, pero debemos añadir esos parámetros a self como se ve en las últimas líneas.

La palabra reservada **raise** sirve para generar un error en cualquier punto del código.



<https://docs.python.org/es/3/tutorial/errors.html#user-defined-excep>