TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- ¿Qué es una excepción?
- Tipos de excepciones
- Errores de sintaxis
- Manejo de excepciones: Try/except y captura de múltiples excepciones

¿QUÉ ES UNA EXCEPCIÓN?

Python utiliza un objeto especial llamado excepción para controlar cualquier error que pueda ocurrir durante la ejecución de un programa.

Una declaración o expresión estando sintácticamente correcta, puede generar un error cuando se intenta ejecutar. Los errores detectados durante la ejecución se llaman *excepciones*.

Cuando ocurre un error durante la ejecución de un programa, Python crea una excepción. Si esta no se controla, la ejecución del programa se detiene y se muestra el error (*traceback*).

```
>>> print(1 / 0)  # Error al intentar dividir por 0.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

TIPOS DE EXCEPCIONES

Veamos los siguientes ejemplos:

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
TypeError: can only concatenate str (not "int") to str
```

La última línea de los mensajes de error indica qué ha sucedido. Hay excepciones de diferentes tipos, y el tipo se imprime como parte del mensaje. En el ejemplo son: `ZeroDivisionError`, `NameError` y `TypeError`. La cadena mostrada como tipo de la excepción es el nombre de la excepción predefinida que ha ocurrido. Esto es válido para todas las excepciones predefinidas del intérprete, pero no tiene por qué ser así para las definidas por el usuario (aunque es una convención útil). Los nombres de las excepciones estándar son identificadores incorporados al intérprete (no son palabras clave reservadas).

El resto de la línea provee información basada en el tipo de la excepción, y qué la causó.

La parte anterior del mensaje de error muestra el contexto donde ocurrió la excepción, en forma de seguimiento de pila. Generalmente contiene un seguimiento de pila que enumera las líneas de origen; sin embargo, no mostrará las líneas leídas desde la entrada estándar.

De este modo, las principales excepciones definidas en Python son:

- **TypeError**: ocurre cuando se aplica una operación o función a un dato del tipo inapropiado.
- **ZeroDivisionError**: ocurre cuando se intenta dividir por cero.
- **OverflowError**: ocurre cuando un cálculo excede el límite para un tipo de dato numérico.
- **IndexError**: ocurre cuando se intenta acceder a una secuencia con un índice que no existe.
- **KeyError**: ocurre cuando se intenta acceder a un diccionario con una clave que no existe.
- **FileNotFoundError**: ocurre cuando se intenta acceder a un fichero que no existe en la ruta indicada.
- **ImportError**: ocurre cuando falla la importación de un módulo.

ERRORES DE SINTAXIS

También conocidos como errores de interpretación, son quizás el tipo más común cuando no se conoce Python:

```
>>>
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
```

```
SyntaxError: invalid syntax
```

El intérprete reproduce la línea responsable del error, y muestra una pequeña “flecha” que apunta al primer lugar donde éste se detectó. El error ha sido provocado (o al menos detectado) en el elemento que precede a la flecha. En el ejemplo, el error se detecta en la función `print()`, ya que faltan dos puntos (':') antes del mismo. Se muestran el nombre del archivo y el número de línea para que pueda mirar en caso de que la entrada venga de un programa.

MANEJO DE EXCEPCIONES

Para el manejo de excepciones los lenguajes proveen ciertas palabras reservadas, los cuales permiten manejar las excepciones que puedan surgir, y tomar acciones de recuperación para evitar la interrupción del programa o, al menos, para realizar algunas acciones adicionales antes de interrumpirlo.

En el caso de Python, el manejo de excepciones se hace mediante los bloques que utilizan las sentencias *try*, *except* y *finally*.

LA SENTENCIA TRY/EXCEPT

Dentro del bloque ***try*** se ubica todo el código que pueda llegar a levantar una excepción, y se utiliza el término levantar para referirse a la acción de generar una excepción.

A continuación, se ubica el bloque ***except***, el cual se encarga de capturar la excepción, y nos da la oportunidad de procesarla mostrando, por ejemplo, un mensaje adecuado al usuario. Veamos qué sucede si se quiere realizar una división por cero:

```
>>> dividendo = 5
>>> divisor = 0
>>> dividendo / divisor
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

En este caso, se levantó la excepción **ZeroDivisionError** cuando se quiso hacer la división. Para evitar que se levante la excepción, y se detenga la ejecución del programa, se utiliza el bloque **try-except**.

```
>>> try:
...     cociente = dividendo / divisor
... except:
...     print "No se permite la división por cero"
...
No se permite la división por cero
```

CAPTURA DE MÚLTIPLES EXCEPCIONES

Dado que dentro de un mismo bloque ***try*** pueden producirse excepciones de distinto tipo, es posible utilizar varios bloques ***except***, cada uno para capturar un tipo distinto de excepción.

Esto se hace especificando, a continuación de la sentencia ***except***, el nombre de la excepción que se pretende capturar. Un mismo bloque ***except*** puede atrapar varios tipos de excepciones, lo cual se hace especificando los nombres de las excepciones separados por comas luego de la palabra ***except***. Es importante destacar que, si bien luego de un bloque ***try*** puede haber varios bloques ***except***, se ejecutará, a lo sumo, uno de ellos.

```
try:
    # aquí ponemos el código que puede lanzar excepciones
except IOError:
    # entrará aquí en caso de que se haya producido
    # una excepción IOError
except ZeroDivisionError:
    # entrará aquí en caso de que se haya producido
    # una excepción ZeroDivisionError
except:
    # entrará aquí en caso de que se haya producido
    # una excepción que no corresponda a ninguno
    # de los tipos especificados en los except previos
```

Como se muestra en el ejemplo precedente, también es posible utilizar una sentencia ***except*** sin especificar el tipo de excepción a capturar. En ese caso, se captura cualquier excepción, sin importar su tipo. Cabe destacar, también, que en caso de utilizar una sentencia ***except*** sin especificar el tipo, la misma debe ser siempre la última de las sentencias ***except***, es decir, que el siguiente fragmento de código es incorrecto.

```
try:
    # aquí ponemos el código que puede lanzar excepciones
except:
    # ERROR de sintaxis, esta sentencia no puede estar aquí,
    # sino que debería estar luego del except IOError.
```

```
except IOError:
    # Manejo de la excepción de entrada/salida
```