

Excepciones.

Karina Flores G.

Programa de Tecnología en Cómputo.



pythonTM



Tipos de errores.


En los lenguajes de programación existen dos tipos de errores:

- **Error de sintaxis:** Ocurren cuando dentro del código fuente existe una sentencia o bloque de código que no cumple con la sintaxis del lenguaje y por lo tanto no puede interpretarse.

```
>>> while True print 'Hola mundo'
...
    while True print 'Hola mundo'
                ^
SyntaxError: invalid syntax
```



- **Excepciones:** Ocurren cuando se produce un error durante la ejecución aunque la sintaxis sea correcta. La mayoría de las excepciones no son manejadas por los programas y son mostrados como errores en el código.



```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero

>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined

>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```



Todas las excepciones son clasificadas en tipos, el cual se muestra al final del mensaje de error. Existen muchas excepciones predefinidas por el intérprete de python, estas no son palabras reservadas y pueden ser consultadas en la documentación. Algunos ejemplos de excepciones son:

- ▶ ImportError
- ▶ NameError
- ▶ IndexError
- ▶ ModuleNotFoundError

Sin embargo el usuario puede definir sus propias excepciones con el objetivo de que el intérprete puede detectarlas como error y darles un manejo especial.



Manejo de excepciones

Es posible escribir código para poder manejar las excepciones, es decir que en caso de que se presente una excepción la ejecución no se interrumpa y en su lugar se ejecute algún código.

Para poder hacer manejo de una excepción es necesario hacer uso de las palabras reservadas **try** y **except**.

```
>>> while True:
...     try:
...         x = int(raw_input(u"Por favor ingrese un número: "))
...         break
...     except ValueError:
...         print u"Oops! No era válido. Intente nuevamente..."
... 
```



¿Cómo funciona?

- ▶ **Try:** El bloque de código que se encuentre indentado respecto al try será lo que siempre se ejecute, en caso de que se presente alguna excepción en la ejecución de este código se procede a la sentencia except.
- ▶ **Except:** Aquí es en donde se maneja la excepción. Después de la palabra reservada except se especifica el tipo de excepción que se desea cachar. Si el nombre de la excepción que el código lanza coincide con la que se especifico, entonces se ejecuta el bloque de código indentado respecto al except.



Observaciones

- ▶ Un solo try puede contener varios except para poder manejar distintos tipos de excepciones.
- ▶ Siempre se ejecuta uno y solo un manejador.
- ▶ Un except puede nombrar muchas excepciones haciendo uso de paréntesis para especificarlas.

```
... except (RuntimeError, TypeError, NameError) :  
...     pass
```



- El último except puede no especificar alguna excepción lo que puede servir para abarcar todas las que no fueron nombrada anteriormente. Pero **¡MUCHO CUIDADO!** esto puede no resultar factible debido a que puede evitar que nos muestre un error real y muy importante.

```
import sys

try:
    f = open('miarchivo.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print "Error E/S ({0}): {1}".format(errno, strerror)
except ValueError:
    print "No pude convertir el dato a un entero."
except:
    print "Error inesperado:", sys.exc_info()[0]
    raise
```



Else

Los try-except cuentan con otro bloque llamado **else**, este es usado para el bloque de código que debe ejecutarse en caso de que el try no genere una excepción.

```
try:
    f = open(arg, 'r')
except IOError:
    print 'no pude abrir', arg
else:
    print arg, 'tiene', len(f.readlines()), 'lineas'
    f.close()
```

NOTA: Se debe procurar que el bloque try unicamente incluya el código en el que es posible encontrar una excepción y se desea manejar.



Finally

Existe otra palabra reservada que puede ser utilizada con los bloques try, esta es **finally**. Esta cláusula es opcional y su característica es que siempre es ejecutada, sin importar si el try generó una excepción o no.

```
>>> try:
...     raise KeyboardInterrupt
... finally:
...     print 'Chau, mundo!'
...
Chau, mundo!
KeyboardInterrupt
```



Levantamiento de excepciones

También es posible obligar al código a que genere una excepción de algún tipo en un momento específico. Para levantar una excepción es necesario utilizar la palabra reservada **raise**.

```
>>> raise NameError('Hola')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: Hola
```



Creación de excepciones

El usuario puede crear sus propias excepciones para el código, es decir que puede obligar al programa a que ocurra una excepción cuando se realiza una acción, sin importar cual sea.

La creación de excepciones serán manejadas con una clase, dicha clase heredará de la clase Exception.

```
>>> class MiError(Exception):  
...     def __init__(self, valor):  
...         self.valor = valor  
...     def __str__(self):  
...         return repr(self.valor)  
...  
>>> try:  
...     raise MiError(2*2)  
... except MiError as e:  
...     print u'Ocurrió mi excepción, valor:', e.valor  
...  
...
```



TAREA 1 :)

Realizar una calculadora con las operaciones básicas: suma, resta, multiplicación y división.

Para el caso de la suma, la resta y la multiplicación deberá manejarse una excepción para que el usuario no pueda ingresar cadenas, solo números. Para el caso de la división debe manejarse la excepción anterior y no permitir las divisiones entre cero. Por último agregar una última operación que permita saber si un número es múltiplo de otro, en caso de no serlo, crear su propia excepción y lanzarla para advertirle al usuario que no son múltiplos.

En este caso, es necesario que el primero número que se ingrese se considere como el múltiplo del segundo, no al revés.



TAREA 2 :)

Crear un sistema de loggeo para que un usuario pueda acceder a un sistema. Como en cualquier login se le solicita su nombre de usuario y contraseña estás deben ser definidas en el programa o pueden almacenarse en un archivo (puntos extras). Si la contraseña no es correcta lanzar una excepción llamada **PasswordSystemError** creada por ustedes, en caso de que sea correcta mostrar un mensaje e bienvenida.



Referencias.



 CEBALLOS, F. J. (2004). ENCICLOPEDIA DEL LENGUAJE C. MÉXICO: ALFAOMEGA/RAMA.

 GUIDO, R.. (2009). EL TUTORIAL DE PYTHON PSF.

