Python 3 para impacientes

"Simple es mejor que complejo" (Tim Peters)

sábado, 1 de febrero de 2014

Excepciones

Las **excepciones** o errores que se pueden producir durante la ejecución de un programa Python se gestionan con una construcción **try-except**. Una excepción puede ocurrir al intentar operar con datos de distinto tipo, dividir un número por cero, teclear un tipo de dato no esperado, acceder a un archivo que no existe y en otras situaciones previsibles. Para cuando sucedan es importante decidir qué respuesta dar en cada caso y evitar que el programa suspenda su ejecución de manera inesperada.

En el bloque **try** se incluye el código que puede producir la excepción y en **except** el código que la tratará convenientemente. En un bloque **except TipoError** se capturará un tipo de error concreto, que habrá que indicarlo. Por ejemplo, si se intenta convertir una cadena de texto de caracteres alfabéticos a número entero el tipo es **ValueError** o si se opera con una variable no declarada el tipo es **NameError**.

try: # contiene el código que puede producir la excepción
except: # contiene el código que gestionará cualquier error
except TipoError: # contiene el código que gestionará un tipo de error concreto
else: # contiene el código que se ejecutará en caso de no haber error
finally: # contiene el código que se ejecutará en cualquier caso.

El siguiente ejemplo calcula el factorial de un número. Si no se introduce un número entero se producirá una excepción:

```
try:
    numero = int(input('Introducir un número: '))
    factorial = 1
    for num in range(1, numero+1):
        factorial *= num

    print(factorial)

except:
    print('Debe introducir un número entero')
```

A continuación, se utilizan el resto de clausulas disponibles para la gestión de excepciones.

```
try:  # Bloque de código a "vigilar"
    texto = input('Teclear :')  # introducir un dato

except KeyboardInterrupt:  # captura excepción de interrupción
    print('\nSe ha pulsado ctrl+c') # Interrupción al presionar Ctrl+c

else:  # se ejecuta si no hay error
    print('Ha tecleado {}'.format(texto)) # muestra cadena introducida

finally:  # se ejecuta tanto si hay error como si no
    print('fin de programa')  # muestra mensaje final
```

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Convertir, copiar, ordenar, unir y dividir arrays Numpy
Esta entrada trata sobre algunos métodos que se utilizan en Numpy para convertir listas en arrays y viceversa; para copiar arrays d...

Tkinter: Tipos de ventanas
Ventanas de aplicación y de diálogos En la entrada anterior tratamos los distintos gestores de geometría que se utilizan para di...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones

Excepciones a medida

La flexibilidad en la gestión de excepciones se amplia cuando los programadores pueden definir sus propios tipos de errores. Para ello, es necesario crear una clase que herede de la clase **Exception** o de alguna de sus hijas y después llamarla con **raise**:

```
# Define clase a partir de Exception
class LongPassw(Exception):
    '''Excepción definida por usuario'''
    def __init__(self, longitud):  # define método constructor ...
        Exception.__init__(self)  # … de excepción ...
        self.longitud = longitud  # … y con atributo longitud

try:  # bloque de código a vigilar
    clave = input('Teclear contraseña: ')  # introducir una cadena
    if len(clave) < 6:  # si longitud de cadena es menor de 6
        raise LongPassw(len(clave))  # llama a excepción de usuario

except LongPassw as lp:  # excepción de usuario
    print('LongPassw: Error por longitud: {0}'.format(lp.longitud))

else:  # se ejecuta si no hay error
    print('No se ha producido error.')  # muestra mensaje
```

De obligado cumplimiento: assert

Con **assert** se fijan las condiciones que debe cumplir un objeto y si éstas no se cumplen en un momento determinado, producirá una excepción que hay que interceptar y tratar convenientemente.

En el siguiente ejemplo se define una lista con seis elementos. 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 dos elementos. Cuando haya dos y se intente borrar de nuevo se producirá la excepción **AssertionError**.

```
try:  # bloque de código a vigilar
    lista = [1, 2, 3, 4, 5, 6]  # define lista
    print(lista)  # muestra lista
    while True:  # bucle infinito hasta error
        print('Elemento a borrar',lista[-1])
        lista.pop()  # borra elemento
        assert len(lista) > 1  # la lista debe tener al menos 2
        print(lista)  # muestra lista después de borrado

except AssertionError:  # excepción para assert
    print('Error al intentar borrar elemento  ')  # mensaje de assert
    print('La lista debe contener al menos 2')
```

Ir al índice del tutorial de Python

Publicado por Pherkad en 3:20

Etiquetas: Python3

Entrada más reciente Inicio Entrada antigua

que permiten obtener de distintos modos números a...

Archivo

febrero 2014 (17)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears