Soporte a la Gestión de Datos con Programación Visual

MANEJO DE ERRORES Y EXCEPCIONES

OBJETIVO

- Para el usuario:
  - Recibir un mensaje comprensible y opciones claras
- Para el desarrollador:
  - Tener información lo más completa posible sobre lo ocurrido
  - Manejar algunas situaciones previsibles sin que el programa se cierre.

ERRORES Y EXCEPCIONES

Errores de sintaxis:

Por ej.:

```
def funcion1( ..)        #<<<  faltan los :
        sentencias …
```

Genera un mensaje de SyntaxError: invalid syntax

Aun cuando la sintaxis esté correcta, puede surgir un error al momento de la ejecución, por ej. al querer dividir un nro por cero. Esto genera una ***Excepción***:

Aparece un mensaje Traceback … (línea del error) ….

ZerodivisionError

EXCEPCIONES

- Alteran el flujo de ejecución del programa.
- Pueden ser interceptadas y también generadas.
- try / except Permiten interceptarlas
- raise la genera manualmente
- Se usan para:
  - Manejo de errores
  - Notificar eventos (evita una cadena de banderas de resultado. Por ej, una rutina de búsqueda lanza una excepción en lugar de devolver 1, en caso de éxito)
  - Manejo de casos especiales
  - Control de flujo inusual (las excepciones son un caso de goto de alto nivel)
- Una excepción es un objeto, de clases que heredan de Exception (que a su vez hereda de BaseException).

MANEJO DE LAS EXCEPCIONES

Pueden manejarse varios errores previsibles y uno genérico también. Por ej.:

```
import sys

try:
    f = open('miarchivo.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("Error del Sist.Operativo: {0}".format(err))
except ValueError:
    print("No se pudo convertir a entero el dato leído.")
except:
    print("Error inesperado:", sys.exc_info()[0])
    raise
else:
    print("El archivo se leyó correctamente")
finally:
    f.close()                    #--- siempre se cierra
```

La expresión raise permite que, ante un error inesperado, sea manejado por quien llamó a este bloque de código. Raise sin parámetros, genera nuevamente la excepción vigente.

TIPOS DE ERRORES HABITUALES

IOError Si un archivo no se puede abrir.

ImportError Si Python no encuentra un módulo

ValueError Cuando una operación o función recibe un valor inapropiado

KeyboardInterrupt Cuando el usuario presiona tecla de interrupción: (Control-C o Delete)

EOFError Cuando al leer datos de un archivo (input() or raw_input()) encuentra final de archivo (EOF) sin leer datos.

MANEJO DE UN ERROR ESPECIFICO

```
def esto_falla():
        x = 1/0

try:
    esto_falla()
except ZeroDivisionError as err:
    print('Manejamos error de División por cero:', err)
```

Otro ejemplo:
Esperamos que el usuario escriba un nro, y validamos un tipo específico de error: *de valor*.

```
while True:
    try:
        x = int(input("Escriba un nro:"))
        break
    except ValueError:
        print("No es un número!. Intente de nuevo")
```

ELSE Y FINALLY

Cláusulas **else** y **finally**:
Ejemplo:

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:          # --- si se da error específico
        print("división por cero!")
    else:                              #--- si no hubo errores:
        print(“el resultado es: ", result)
    finally:                           #--- se ejecuta siempre
        print(“para limpiar…")
```

EXCEPCIONES GENERADAS POR EL USUARIO

Es posible generar excepciones. Se suelen emplear para enviar mensajes entre capas. Hay que asignar un identificador y un texto

```
class MiError(Exception):                    #--- debe ser subclase de Exception
   pass                                      #--- en este caso no modificamos comportamiento

def conectar():
    try:
        if ….:              #--- no hay conexión con b.datos por ej
        raise MiError('No nos conectamos con la b.datos!')          #--- genera instancia de MiError
```

En este caso se genera un error, que debería ser manejado por quien llamó a esta parte de código, mediante:

```
try:
    conectar():
except MiError as e:
    print ('Error: ', e.args)                # mostrará el mensaje 'No nos conectamos ….'
```

RESUMEN TRY - EXCEPT

Cláusula	Interpretación
except:	Captura todos (los otros) tipos de excepción
except nombre:	Captura una excepción específica
except nombre, valor:	Captura excepción y sus datos extra
except(nom1, nom2)	Captura cualquiera de las excepciones listadas
else:	Bloque que se ejecuta si no se disparó excepción
finally:	Bloque que se ejecuta siempre.

TRYS ANIDADOS

```
def accion2():
    print (1 + [] )                 #--- genera TypeError

def accion1 () :
    try:
        accion2()
    except TypeError:               #--- este tray es el más cercano al error
        print (Excepción disparada en accion1()‘)

try:
    accion1()
except TypeError:                   # --- llega aquí sólo si accion1 levanta de nuevo el error
    print (‘Excepción disparada en nivel superior‘)
```

Al ejecutar, obtenemos: excepcion de accion1

El primer try que toma un error, hace desaparecer la excepción a menos que llame a raise. Probar.

POSIBLE ESTRATEGIA USANDO EXCEPCIONES COMO REEMPLAZO DE BANDERAS

```
def hace_algo():
    primer_paso()       #--- Aquí no tenemos en cuenta los posibles
    segundo_paso()      #--- errores. Seguimos el caso más favorable
    ...
    ultimo_paso()


try:
    hace_algo()         #--- Aquí sí tenemos en cuenta los errores
except:                 #--- Es el único lugar, con lo que se simplifica
    mal_final()         #--- la programación
else:
    buen_final()
```

LEVANTAR EXCEPCIONES

raise nombre #--- identifica la excepción con un nombre, para el except que lo manejará

raise nombre, datos #--- agrega datos opcionales (por defecto, None)

raise clase, instancia #--- except

raise instancia #--- similar anterior

raise #--- sin parámetros, levanta la excepción vigente

USO BÁSICO

Ejemplo

```
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
   LAST_NAME, AGE, SEX, INCOME)
   VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""

try:
   #--- Ejecutar comando
   cursor.execute(sql)
except:
   #--- si hubo algún error - rollback
   db.rollback()
else:
   #--- grabar cambios
   db.commit()
```

OTRO EJEMPLO

```
try:
    conn = MySQLConnection(**db_config)

    # --- actualizar datos
    cursor = conn.cursor()
    cursor.execute(query, data)

    # grabar cambios
    conn.commit()

except Error as e:          #--- genérico – para cualquier error
    print(e)

finally:                    #--- cerrar conexión acá. Se ejecuta siempre
    cursor.close()
    conn.close()
```

ASSERT

Assert dispara una excepción del tipo:

AssertionError si la cláusula es falsa.

assert expresión [, argumentos]

Ej:

```
def divide( a, b ):
        assert ( b != 0 ), ‘divisor distinto que cero!!’
        return (a/b)

print( divide ( 4, 0 ))
```

Devuelve:

AssertionError: divisor distinto que cero!!

EJERCICIOS

1) escribir una función que reciba 2 números, haga el cociente y devuelva el resultado

Que sea robusta, que admita división por 0 o que le pasen un texto.

2) Usar una excepción propia.

Escribir una función que genere esa excepción y sea captada por quien la llama.

3) Mejorar las funciones de gestión de datos con Mysql, capturando algunos errores (generarlos a propósito).