






Control de errores

- Los errores en los programas aparecen por diferentes situaciones, se pueden dar por un código mal programado, unos datos mal introducidos, o por otro tipo de causas.
- Independientemente de por qué se ha generado, es imprescindible realizar una gestión adecuada del fallo para evitar que la aplicación finalice sin ninguna explicación.
- Para la adecuada gestión de los errores, el sistema nos va a presentar un modelo basado en la gestión de **excepciones**.
- Una **excepción** es un objeto que contiene información sobre un error que se ha producido durante la ejecución.

- 
- Toda **excepción** generada tendrá que recogerse y tratarse de alguna manera, si no el propio intérprete proporciona un mecanismo por defecto que consiste en mostrar un mensaje por consola y parar la aplicación.
 - Las **excepciones** son objetos explicativos, siendo una de las informaciones más valiosas que aportan el tipo.
 - Cada una tendrá un tipo definido en el lenguaje dándonos datos sobre las causas que han generado el fallo.
 - Así, una **excepción** de **IOError** determinará un error en una operación de entrada - salida, una de **MemoryError** se generará cuando no quede memoria suficiente, etc.

- 
- Las **excepciones** se lanzan por el sistema y estamos en la obligación de capturarlas evitando el comportamiento por defecto.
 - La gestión se realiza mediante una construcción conocida de otros lenguajes: **try...except...else...finally** (recuerda el **try...catch** de **Java**).
 - Para capturar un posible error encerraremos dentro del bloque **try** las sentencias susceptibles de generarlo, determinando con tantas sentencias **except** como necesitemos los posibles tipos de error producidos, si dos errores se van a tratar de forma similar es posible añadir en una línea **except** ambos, encerrándolos entre paréntesis y separándolos con comas.

- 
- Añadiremos un bloque **else** si queremos incluir un tratamiento por defecto en caso que no se produzca un error y dentro de **finally** se situarán aquellas sentencias que se ejecutarán siempre tras el tratamiento de un problema independientemente de este, generalmente para realizar algún tipo de limpieza.
 - Hemos dicho que el sistema genera las **excepciones**, pero en algunos casos es recomendable usar nosotros el mismo mecanismo para comunicar un error.
 - Podemos lanzar una excepción en cualquier momento con la palabra clave **raise** y un objeto que herede de **Exception**. Dotando de esta manera un mecanismo coherente de gestión de los errores independientemente de cómo se produzcan.



➤ Las excepciones más comunes son:

- **BaseException.** Clase base para todas las demás.
- **Exception.** Clase base para todas las excepciones que no son de entrada - salida.
- **AithmeticError.** Clase base para los errores aritméticos, extienden de ella: `FloatingPointError`, `OverflowError`, `ZeroDivisionError`.
- **EOFError.** Sobrepasado el fin del fichero.
- **IOError.** Error de entrada - salida.
- **ImportError.** Error en la importación, con este error podemos comprobar funcionalidades instaladas en el sistema.
- **IndexError.** Error de índice en el acceso.
- **KeyError.** Clave no existente.

Actividad 5.28

Crea un proyecto llamado **control de errores** y un módulo **Actividad5-28**

```
# -*- coding: utf-8 -*-
try:
    236/0
except NameError:
    print "Error, la variable no existe"
except (ValueError, OverflowError):
    print "Error de valor u overflow"
except Exception as ex:
    print "Error:", ex
else:
    print "Si no hay excepción"
finally:
    print "Continuando..."
class miExcepcion(Exception):
    def __init__(self, val):
        self.valor = val
    def __str__(self):
        return "Error:" + str(self.valor)
try:
    raise miExcepcion("Prueba")
except miExcepcion, e:
    print e
```


Console


```
<terminated> C:\Users\Web\workspace\Control de errores\src\Actividad5-28\Actividad5-28.py
Error: integer division or modulo by zero
Continuando...
Error:Prueba
```



Ficheros

- No es normal que tengamos que crear ficheros o tratarlos de forma manual dentro del sistema **OpenERP** ya que disponemos de una base de datos.
- La primera operación necesaria para acceder a un fichero es su apertura, realizándose esta mediante la instrucción **open(fichero, modo)**.
- En el primer parámetro determinamos el fichero a abrir incluyendo la ruta.
- El segundo parámetro nos indicará el modo en el que tenemos que abrir dicho fichero (**r** para lectura, **w** para escritura, **a** para añadir y **b** en modo binario).

- 
- Esta operación es sensible y puede lanzar diversas **excepciones** por lo que es recomendable capturarla dentro de un bloque **try**.
 - Si se realiza, la acción nos devolverá un objeto que usaremos para el acceso.
 - Tras la apertura y el tratamiento tenemos obligatoriamente que liberar las estructuras de memoria y grabar los cambios a disco. Esta acción la realizaremos a través de la llamada a **close()** del objeto devuelto.
 - Para gestión general de los datos nos encontramos multitud de métodos a través del objeto que devuelve la sentencia **open**:

- 
- Para escribir en disco tenemos la función **write()**,
 - si necesitamos leer datos usaremos **read(numBytes[,MaxBytes])** o **readline()**,
 - para determinar la posición actual en el fichero existe el método **tell()**,
 - Para posicionarlo en un lugar concreto se utiliza **seek(cuanto,cómo)** indicado el parámetro **cómo** el mecanismo a usar para contar los bytes indicados en el primer parámetro (los valores posibles son 0 para indicar un valor absoluto desde el principio, 1 para indicar un desplazamiento relativo desde la posición actual y 2 para interpretarlo como un desplazamiento relativo desde el final del fichero).

Actividad 5.29

Crea un proyecto llamado **Ficheros** y un módulo **Actividad5-29** con el código siguiente. Ejecútalo.

```
# -*- coding: utf-8 -*-
try:
    f = open("mifichero2", "a+")
except Exception, ex:
    exit

f.write("hola\r")
f.close()

f = open("mifichero2", "r")
while True:
    print "Posición:", f.tell()
    I = f.readline()
    if not I: break
    print I,

print "Esta cerrado?", f.closed
f.close()
print "Esta cerrado?", f.closed

# otra forma no es necesario cerrar
with open("mifichero2") as archivo:
    for linea in archivo:
        print linea
```


Console


```
<terminated> C:\Users\Web\workspace\Ficheros\src\Actividad5-29\A
Posición: 0
hola
Posición: 5
Esta cerrado? False
Esta cerrado? True
hola
```



Módulos SYS e IO

- Dentro de los módulos disponibles en **Python**, hay dos que son de vital importancia el conocerlos y tratarlos.
- El módulo **SYS** : encargado de la información relativa al sistema en el que se encuentra el intérprete.
- El módulo **IO**: se utiliza para la gestión a bajo nivel de ficheros y directorios.
- El módulo **SYS** implementa funciones para recabar información acerca del sistema en el que nos encontramos.

- 
- Dentro de las funciones más usadas encontramos una para determinar la versión del API del motor **Python** que estamos usando (**sys.api_version**), la obtención de la codificación de los caracteres por defecto (**sys.getdefaultencoding()**), el **path** actual (**sys.path**) o la plataforma (**sys.platform**) en la que se está corriendo el script.
 - Para el módulo **IO**, las funciones más interesantes son **os.getcwd()** si necesitamos encontrar el directorio actual de trabajo, **os.environ** para determinar las variables del entorno o **os.getpid()** para conseguir el PID de un proceso, junto con las típicas de gestión de archivos **access**, **chdir**, **mkdir**, **open**, **remove**, **read**, **rename**.
 - Una función muy útil es el método **os.listdir(path)** que nos proporcionará una lista con los contenidos del directorio que le pasemos como parámetro. Lista que recorreremos de forma sencilla con un bucle **for in**.

- 
- La función **listdir** plantea un problema, no accede de forma automática a los directorios contenidos dentro de la ruta que le pasamos. Este problema se solventa llamado al método **os.walk(ruta)** que recorrerá de forma recursiva toda la estructura desde el punto que le indiquemos, devolviendo una lista con tres valores, el **path**, los directorios y todos los ficheros.
 - Importante es también el objeto **os.path** que presenta funciones de gestión de rutas necesarias en la programación. Este objeto incorpora el método **abspath()** que devuelve la ruta absoluta de aquella que le pasemos, funciones para dividir una ruta en la parte de fichero (**dirname**) y en la parte de ruta (**basename**), etc.
 - En aquellos casos que necesitemos terminar nuestro script sin haber llegado al punto final, podemos llamar a la función **sys.exit()**.

Actividad 5.30

Crea un proyecto llamado **SYS_IO** y un módulo **Actividad5-30** con el código siguiente. Ejecútalo.


```
# -*- coding: utf-8 -*-
import sys, os


print "API:", sys.api_version
print "Encode:", sys.getdefaultencoding()
print "Tamaño de Hola:", sys.getsizeof("Hola", 0)
print "Path:", sys.path
print "Plataforma:", sys.platform
print "Versión:", sys.version
print sys.modules
print "Módulo os gestión archivos y directorios"
print "Directorio actual:", os.getcwd()
print "Path:", os.defpath
print "Variables de entorno:", os.environ
print "Pid:", os.getpid()
# access, chmod, mkdir, open, remove, read, rename, rmdir, dup, dup2, exec
for fil in os.listdir(os.getcwd()):
    print "Nombre:", fil
for path, dirs, files in os.walk(os.getcwd()): # Para cada directorio de forma
    for fil in files: # recorremos todos los ficheros encontrados
        filename = os.path.join(path, fil)
        print "Tamaño de " + filename + " es ", os.path.getsize(filename)
# os.path:
path = os.getcwd()
print "Objeto os.path"
print "path absoluto:", os.path.abspath(path)
print "basename:", os.path.basename(path)
print "dirname:", os.path.dirname(filename)
print "tiempo de acceso:", os.path.getatime(path)
print "es un directorio:", os.path.isdir(path) # isfile, islink, ismount
print "la unidad es:", os.path.splitdrive(path)[0]
sys.exit()
```


API: 1013
Encode: ascii
Tamaño de Hola: 25
Path: ['C:\\Users\\Web\\workspace\\SYS_IO\\src\\Actividad5-30',
Plataforma: win32
Versión: 2.7.11 (v2.7.11:6d1b6a68f775, Dec 5 2015, 20:32:19) [M
{'copy_reg': <module 'copy_reg' from 'C:\\Python27\\lib\\copy_reg.p
Módulo os gestión archivos y directorios
Directorio actual: C:\\Users\\Web\\workspace\\SYS_IO\\src\\Actividad5-
Path: .;C:\\bin
Variables de entorno: {'TMP': 'C:\\Users\\Web\\AppData\\Local\\T
Pid: 5876
Nombre: Actividad5-30.py
Nombre: __init__.py
Tamaño de C:\\Users\\Web\\workspace\\SYS_IO\\src\\Actividad5-30\\Activi
Tamaño de C:\\Users\\Web\\workspace\\SYS_IO\\src\\Actividad5-30__init
Objeto os.path
path absoluto: C:\\Users\\Web\\workspace\\SYS_IO\\src\\Actividad5-30
basename: Actividad5-30
dirname: C:\\Users\\Web\\workspace\\SYS_IO\\src\\Actividad5-30
tiempo de acceso: 1454509812.84
es un directorio: True
la unidad es: C:

Expresiones regulares


- Las expresiones regulares son cadenas de texto que formarán un patrón de búsqueda indicando concordancia o no con el patrón. Básicamente es un texto que determinará la forma exacta que tiene que tener otro para cumplir con el patrón.
- Se utilizan siempre que necesitemos determinar si una cadena de texto cumple un formato preciso.
- Supongamos, por ejemplo, una aplicación que recoge los datos de personas y necesitamos el D.N.I. Tenemos dos opciones, confiar en que el usuario no se va a equivocar u olvidar de introducir la letra, o confirmar que se cumple un patrón en donde después de una serie de números hay una letra (otra cuestión diferente es que la letra concuerde con el número).

- 
- Para comenzar, indicaremos que vamos a hacer uso de las expresiones regulares importando el módulo correspondiente con la sentencia **import re**.
 - A continuación, crearemos una cadena de texto que será nuestro patrón (la expresión regular), esta cadena contendrá algunos caracteres especiales que tendrán un significado diferente. Por ejemplo: `reExpresion = "abe"`. En este caso estaríamos buscando la cadena «abe».
 - El último paso es determinar si una cadena cualquiera cumple el patrón usando algunas de las funciones a nuestra disposición devolviendo verdadero si cumple el patrón y falso en caso de que no lo cumpla. Por ejemplo: `re.match(reExpresion, "Abe")`. La cadena no cumple el patrón por la «A» mayúscula que en el patrón es «a» minúscula.

- 
- Como podemos ver el mecanismo de uso es sencillo, existen algunas funciones más (**groups()** devuelve los grupos encontrados por la expresión, **sub()** sustituye un valor por el que coincide) y propiedades (se puede determinar que ignore mayúsculas y minúsculas, etc.) que podemos usar pero básicamente es siempre lo mismo: definimos la expresión y la comprobamos.
 - El verdadero problema de las expresiones regulares es la creación y utilización de los caracteres **comodines**, abordaremos ahora este problema. Una expresión regular puede estar formada por:
 - Un conjunto de caracteres que corresponderá exactamente con dicho literal.
Ejemplo: `expReg="abc"` coincidirá con "abe" pero no con "Abe" o "aBc" o "abed".

- 
- El carácter punto (.) casará con cualquier carácter menos el vacío y el salto de línea. Ejemplo: `expReg="a.c"` concordará con `"aBc"` pero no con `"Abe"` o `"ac"` o `"abcd "`.
 - La estructura `(valor1|valor2|..|valorn)` coincidirá con cualquiera de los valores exactamente. Ejemplo: `expReg="(ab|cd)"` se ajustará con `"ab"` y con `"cd"` pero con ninguno más.
 - Los corchetes se utilizan para sustituir un carácter por cualquiera de los indicados dentro. Se pueden escribir secuencias de caracteres o rangos mediante el uso del guión (-). Ejemplo: `expReg="a[123d-g]c"` coincidirá con `"a1c"` y con `"aec"` pero con ninguno de longitud dos o cuatro o más. La expresión pide que empiece por a, a continuación uno de los valores 1, 2, 3, d, e, f, g y por último una c.

- Para indicar ninguno de los valores dentro de los corchetes se antepone el carácter “^”. Ejemplo: `expReg="a[^123d-g]c"` corresponderá a palabras de longitud tres donde su segundo carácter no sea 1,2,3,d,e,f o g.
- Forzar el patrón al comienzo de la frase se hace mediante “^”, pero si lo necesitamos al final de la línea será “\$”. Ejemplo: `expReg="^abc"`, al principio de la frase empezará obligatoriamente por “abc”; `expReg="abc$"`, termina obligatoriamente en “abc”.
- Para buscar una concordancia al principio de palabra se utiliza: “\b”. Si queremos que un carácter sea cualquier número “\d” y si es no numérico “\D”. Ejemplo: `expReg="\ba\dc\D"`. Al inicio de palabra debe haber una “a” seguida de cualquier número, más una letra “c” terminando con cualquier carácter no numérico (que no es lo mismo que cualquier letra).

- 
- Existe un conjunto de patrones especiales para controlar la repetición del carácter anterior. Una interrogación (?) indicará que se puede repetir cero o una vez, un asterisco (*) cero o más veces, un signo más (+) una o más veces y las llaves ({}) encerrarán dos valores separados por una coma indicando el mínimo número de veces permitidas y el máximo. Ejemplo `expReg="a.+1{2,3}c"`. La expresión cuadrará con palabras de cinco o más letras que empiecen en “a” y terminen por “c”, con al menos dos unos y tres como máximo obligatoriamente junto a la “c” final.
 - Cuando necesitemos comprobar que aparece uno de los caracteres especiales le antepondremos la barra invertida “\”. Ejemplo `expReg="mifile\\.txt"` concuerda con el nombre: `mifile.txt`.

Actividad 5.31

Crea un proyecto llamado **Expresiones regulares** y un módulo **Actividad5-31** con el código siguiente. Ejecútalo.

```
# -*- coding: utf-8 -*-
import re

reExpresion = "abc"
print "Cadena de caracteres abc"
print "Si" if (re.match(reExpresion, "Abc")) else "No"
print "Si" if (re.match(reExpresion, "abc")) else "No"

reExpresion = ".bc"
print "Punto para cualquier carácter .bc"
print "Si" if (re.match(reExpresion, "Abc")) else "No"
print "Si" if (re.match(reExpresion, "abc")) else "No"

reExpresion = "(abc|Abc)"
print "(|) una de las opciones separadas por |"
print "Si" if (re.match(reExpresion, "Abc")) else "No"
print "Si" if (re.match(reExpresion, "abc")) else "No"
print "Si" if (re.match(reExpresion, "cBc")) else "No"

reExpresion = "[0-9]bc"
print "Corchetes más rango [0-9]bc"
print "Si" if (re.match(reExpresion, "Abc")) else "No"
print "Si" if (re.match(reExpresion, "1bc")) else "No"
print "Si" if (re.match(reExpresion, "7bc")) else "No"

reExpresion = "[^0-9]bc"
print "Corchetes más rango negado [^0-9]bc"
print "Si" if (re.match(reExpresion, "Abc")) else "No"
print "Si" if (re.match(reExpresion, "1bc")) else "No"
print "Si" if (re.match(reExpresion, "7bc")) else "No"


reExpresion = "[0-9a-zA-Z]bc"
print "Corchetes más rango concatenado [0-9a-zA-Z]bc"
print "Si" if (re.match(reExpresion, "Abc")) else "No"
print "Si" if (re.match(reExpresion, "8Bc")) else "No"
print "Si" if (re.match(reExpresion, "x9c")) else "No"
print "Si" if (re.match(reExpresion, "7bc")) else "No"


print "uso de re"
reExpresion = "http://(.+).{2})" # no es necesario escapar dentro de () el.
print "Si" if (re.match(reExpresion, "HTTP://www.boe.es", re.IGNORECASE | re.VERBOSE)) else "No"
print re.match(reExpresion, "http://www.boe.es").groups()
print re.sub("es", "net", "http://www.boe.es")
```

```
Console
<terminated> C:\Users\Web\workspace\Expresiones regulares
Cadena de caracteres abc
No
Si
Punto para cualquier carácter .bc
Si
Si
(|) una de las opciones separadas por |
Si
Si
No
Corchetes más rango [0-9]bc
No
Si
Si
Corchetes más rango negado [^0-9]bc
Si
No
No
Corchetes más rango concatenado [0-9a-zA-Z]bc
No
Si
Si
Si
uso de re
Si
('www.boe.', 'es')
http://www.bonet.net
```

Bases de datos

- Las bases de datos se han vuelto tan comunes que existen implementaciones de sistemas gestores instalados por defecto en los sistemas operativos o en los lenguajes. Este es el caso de **Python** que incorpora el sistema gestor de bases de datos **SQLite** dentro de los paquetes estándar desde la versión 2.5 del intérprete.
- Independientemente de la base de datos incorporada, aparecen librerías para acceder a cualquier tipo de sistema gestor simplemente importando el módulo correspondiente y haciendo uso de las funciones.
- Usaremos **SQLite** para ver un ejemplo básico de uso. La razón es que **OpenERP** raramente utiliza el acceso directo a **PostgreSQL**, en su defecto usa un mecanismo **ORM** (mapeo a objetos de la base de datos) de gestión de los datos integrando todas las tablas en objetos y simplificando su uso.

- 
- Para acceder a una base de datos hay que abrir una conexión con ella (después de importar el módulo correspondiente) facilitándole los datos necesarios para validar el acceso.
 - En el ejemplo que vamos a programar, no hacen falta datos puesto que utilizamos una facilidad presente en **SqLite** que es la de crear las tablas en memoria en vez de hacerlas físicas. Por supuesto cuando termine la aplicación los datos se perderán.
 - Con el objeto recogido si no ha habido ningún error tendremos acceso a todas las funciones.
 - Para acceder a los datos, tanto ejecutando sentencias **SQL** como realizando selecciones sobre las tablas hay que conseguir un cursor para gestionarlos

- 
- Este cursor nos permitirá lanzar sentencias **SQL** de inserción (INSERT), modificación (UPDATE) o borrado (DELETE) como de selección (SELECT), mediante el método (**execute(SentenciaSQL)**).
 - Con SELECT tendremos que recorrer cada uno de los registros conseguidos mediante un bucle **for in** sobre la lista que nos proporciona el método **fetchall()** del cursor. Cada registro será devuelto como una lista que podremos tratar dentro de otro bucle.
 - En la implementación de **SqLite** que estamos usando, los datos no se guardan en la tabla hasta que no se confirmen correctamente. El proceso se puede hacer uno a uno o tras un conjunto de manipulaciones sobre los datos. En cualquier caso es obligatorio llamar al método **commit** de la base de datos para confirmar los cambios.

Actividad 5.32

Crea un proyecto llamado Bases de datos y un módulo Actividad5-32 con el código siguiente. Ejecútalo.

```
# -*- coding: utf-8 -*-
import sqlite3 as dbapi

bbdd = dbapi.connect(":memory:") # Guardar en memoria en vez de en disco

c = bbdd.cursor()

c.execute("""create table em(dni text,nombre text, departamento text)""")
c.execute("""insert into em values ('a1','b1','c1')""")
c.execute("""insert into em values ('a2','b2','c2')""")
c.execute("""insert into em values ('a3','b3','c3')""")
bbdd.commit()

|
c.execute("""select * from em""")
for t in c.fetchall():
    print t

c.execute("""select * from em""")
t = c.fetchone()
while (t):
    print t
    t = c.fetchone()
```

Console

<terminated> C:\Users\Web\workspace\Bases de datos\src\Actividad5-32\Actividad5-32.py

```
(u'a1', u'b1', u'c1')
(u'a2', u'b2', u'c2')
(u'a3', u'b3', u'c3')
(u'a1', u'b1', u'c1')
(u'a2', u'b2', u'c2')
(u'a3', u'b3', u'c3')
```

Documentación

- El documentar nuestro código de forma correcta es una práctica muy beneficiosa tanto para nosotros como programadores o futuros mantenedores de nuestro código.
- **Python** incorpora algunas facilidades nativas de documentación. En primer lugar podemos usar como ya se ha mencionado la almohadilla (#) para hacer un comentario hasta el final de línea, o las triples comillas simples para hacer un comentario multilínea.
- Pero si necesitamos que el intérprete use nuestras anotaciones hay que documentar cada función o clase con una cadena que describa el funcionamiento y componentes necesarios.

- Esta cadena se escribirá obligatoriamente en la línea siguiente de la definición de la función (**def** Mifuncion(..)), en la creación de la clase (**class** Miclase...) o al principio del módulo después de la codificación de caracteres usando la notación de triples comillas.
- El contenido es libre pero se recomienda el uso de las recomendaciones del formato **javadoc**: **@param** indicando un parámetro, **@return** para el valor de retorno, **@author** para el nombre del autor, **@version** para la versión, etc. Ejemplo:

```
# -*- coding: utf-8 -*-  
"""  
    Created el 01/02/2016  
    @author: TERESA  
    """  
def miFun(p):  
    """  
        Mi función con un parámetro  
        @version 1  
        @param p valor necesario  
        @return un texto  
    """  
    if p > 2: return "Mayor de dos"  
    else: return "Menor o igual que dos"
```

Actividad 5.33

Crea un proyecto que utilice la función del ejemplo anterior y documéntalo.

Actividad 5.34

Asigna a una variable la cadena «esto es un ejemplo», y haz que se escriba por pantalla la cadena «un ejemplo».

Actividad 5.35

Convierte la lista $a=[1,[2,[3,4]],5]$ en $[1,[2,[3,4],[6,7]],5]$.

Actividad 5.36

Escribe un programa que utilice un bucle while para crear una lista con los números del 1 al 10 y luego la muestre por pantalla. Repítelo con un bucle for. ¿Cómo lo harías sin usar bucles?

Actividad 5.37

Escribe un programa que, a partir de una tupla cualquiera, obtenga una lista con todos los elementos de la tupla.

Actividad 5.38

Escribe un programa que muestre por pantalla los números múltiplos de 7 entre el 1 y el 1000. Utiliza una función a la que se le pase un número y devuelva True si es múltiplo de 7, o False si no lo es.

Actividad 5.39

En Python, ¿qué diferencia hay entre las asignaciones `a[2] = [6,7]` y `a[2:3] = [6,7]`?