netmind

WeKnowIT

Roche

Python

PROGRAMACIÓN AVANZADA

1. Control de errores, manejo de excepciones
2. Programas multitareas
3. Módulos de la librería estándar
4. Algunas librerías útiles
5. Estructura de proyectos
6. Testing de módulos

1 Control de errores, manejo de excepciones

Control de Errores

- Una de las partes principales de cualquier lenguaje es la posibilidad de gestionar los errores
- Para ello dispone de una estructura Try-Except
- El **try** será el trozo de código que siempre se ejecutará y sólo si falla ejecutará el **except**

Control de Errores

```
while True:
    try:
        n = input("Please enter an integer: ")
        n = int(n)
        break
    except ValueError:
        print("No valid integer! Please try again ...")
print("Great, you successfully entered an integer!")
```

Control de Errores

- Existe la posibilidad de gestionar un finally

```
try:
    x = float(input("Your number: "))
    print(x)
    inverse = 1.0 / x
except ValueError:
    print("You should have given either an int or a float")
except ZeroDivisionError:
    print("Infinity")
finally:
    print("There may or may not have been an exception.")
```

Control de Errores

- Python tiene muchas excepciones integradas que obliga a su programa a generar un error cuando algo en él sale mal.
  - https://www.programiz.com/python-programming/exceptions

- Si se quiere definir una excepción personalizada, simplemente se debe heredar la clase Exception

```
class Error(Exception):
    """Base class for other exceptions"""
    pass
```

Control de Errores

- Asimismo se pueden levantar excepciones con la orde **raise**
  raise excepción(args)

- raise sin parámetros levanta la última exepción

```
# raise exception (args)
try:
    raise ValueError("I have raised an Exception")
except ValueError as exp:
    print("Error", exp)  # Output -> Error I have raised an Exception


# raise execption
try:
    raise ValueError
except ValueError as exp:
    print("Error", exp)     # Output -> Error
```

Gestionando excepciones

- Crea un script que reciba un nombre de archivo como parámetro, verifique si existe y cuente el número de líneas que tiene
- Si el el archivo no existe, debe mostrar un prompt pidiendo un nuevo nombre

2 Programas multitareas

Programas Multitarea

- En Python se pueden manejar distintos hilos paralelos de ejecución
- Existen dos módulos para generar hilos Thread y Threading
- El módulo **Threading** nos facilita la tarea de creación y lanzamiento de hilos de ejecución
  - https://docs.python.org/3/library/threading.html
- El constructor es

```
class threading.Thread(group=None, target=None, name=None, args=(),
kwargs={}, *, daemon=None)
```

  - **group:** debe ser None; reservado para la extensión futura cuando se implementa una clase ThreadGroup.
  - **target:** es el objeto a ser invocado por el método run(). El valor predeterminado es None, lo que significa que no se invoca nada.
  - **name:** es el nombre del subproceso. De forma predeterminada, se construye un nombre único de la forma "Thread-N" donde N es un número decimal.
  - **args:** es la tupla de argumentos para la invocación del target. El valor predeterminado es ().
  - **kwargs:** es un diccionario de argumentos de palabras clave para la invocación del target. El valor predeterminado es {}.

Programas Multitarea

```
import threading


def worker(num):
    """thread worker function"""
    print('Worker {}'.format(num))
    return


threads = []
for i in range(5):
    t = threading.Thread(target=worker, args=(i,))
    threads.append(t)
    t.start()
```

Programas Multitarea

- Métodos de Threading

start()	Inicia la actividad del hilo.
run()	Método que representa la actividad del hilo.
join(timeout=None)	Espere hasta que el hilo termine.
name	Cadena utilizada únicamente con fines de identificación.
ident	El identificador de hilo de este subproceso o Ninguno si el subproceso no se ha iniciado. Se trata de un entero distinto de cero.
is_alive()	Devuelve si el hilo está vivo.
daemon	Un valor booleano que indica si este subproceso es un subproceso de demonio (Verdadero) o no (Falso). Esto se debe establecer antes de que start () se llame, de lo contrario RuntimeError se eleva.

Programas Multitarea

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

import threading
import time

exitFlag = 0


class myThread (threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter

    def run(self):
        print("Starting " + self.name)
        print_time(self.name, self.counter, 5)
        print("Exiting " + self.name)

def print_time(threadName, delay, counter):
    while counter:
        if exitFlag:
            threadName.exit()
        time.sleep(delay)
        print("%s: %s" % (threadName,
time.ctime(time.time())))
        counter -= 1


# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)

# Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print("Exiting Main Thread")
```

Señales entre threads

- Hay momentos en que es importante poder sincronizar las operaciones en dos o más subprocesos.
- Una forma sencilla de comunicarse entre los subprocesos es utilizar objetos Event.
- Un evento gestiona un indicador interno sobre los que los procesos invocantes pueden hacer set() o clear().
- Otros subprocesos pueden esperar wait() a que se establezca el indicador set(), bloqueando el progreso hasta que se permita continuar.

Señales entre threads

```
import logging
import threading
import time

logging.basicConfig(level=logging.DEBUG,
                    format='(%(threadName)-10s)
%(message)s',
                    )


def wait_for_event(e):
    """Wait for the event to be set before doing
anything"""
    logging.debug('wait_for_event starting')
    event_is_set = e.wait()
    logging.debug('event set: %s', event_is_set)


def wait_for_event_timeout(e, t):
    """Wait t seconds and then timeout"""
    while not e.isSet():
        logging.debug('wait_for_event_timeout
starting')
        event_is_set = e.wait(t)
        logging.debug('event set: %s', event_is_set)
        if event_is_set:
            logging.debug('processing event')
        else:
            logging.debug('doing other work')


e = threading.Event()
t1 = threading.Thread(name='block',
                      target=wait_for_event,
                      args=(e,))
t1.start()

t2 = threading.Thread(name='non-block',
                      target=wait_for_event_timeout,
                      args=(e, 2))
t2.start()

logging.debug('Waiting before calling Event.set()')
time.sleep(3)
e.set()
› logging.debug('Event is set')
```

Programas Multitarea

- Más detalles en:
  - https://pymotw.com/3/threading/

Un script multihilos

- Crea un script que descargue las siguientes imágenes usando 4 workers
  - 'https://farm5.staticflickr.com/4117/4787042405_37e548cf3a_o_d.jpg'
  - 'https://farm3.staticflickr.com/2375/2457990042_e6d6982cb2_o_d.jpg'
  - 'https://farm4.staticflickr.com/3149/3104818507_06cf582ba3_o_d.jpg'
  - 'https://farm3.staticflickr.com/2801/4084837185_4c12f32b1f_o_d.jpg'

3 Módulos de la librería estándar

Módulos de sistema

- El módulo os nos permite acceder a funcionalidades dependientes del Sistema Operativo. Sobre todo, aquellas que nos refieren información sobre el entorno del mismo y nos permiten manipular la estructura de directorios (para leer y escribir archivos, ver capítulo 9)
  - http://docs.python.org/library/os.html

Módulos de sistema

Descripción	Método
Saber si se puede acceder a un archivo o directorio	os.access(path, modo_de_acceso)
Conocer el directorio actual	os.getcwd()
Cambiar de directorio de trabajo	os.chdir(nuevo_path)
Cambiar al directorio de trabajo raíz	os.chroot()
Cambiar los permisos de un archivo o directorio	os.chmod(path, permisos)
Cambiar el propietario de un archivo o directorio	os.chown(path, permisos)
Crear un directorio	os.mkdir(path[, modo])
Crear directorios recursivamente	os.mkdirs(path[, modo])
Eliminar un archivo	os.remove(path)
Eliminar un directorio	os.rmdir(path)
Eliminar directorios recursivamente	os.removedirs(path)
Renombrar un archivo	os.rename(actual, nuevo)
Crear un enlace simbólico	os.symlink(path, nombre_destino)

os.path

- Permite acceder a ciertas funcionalidades relacionadas con los nombres de las rutas de archivos y directorios

Descripción	Método
Ruta absoluta	os.path.abspath(path)
Directorio base	os.path.basename(path)
Saber si un directorio existe	os.path.exists(path)
Conocer último acceso a un directorio	os.path.getatime(path)
Conocer tamaño del directorio	os.path.getsize(path)
Saber si una ruta es absoluta	os.path.isabs(path)
Saber si una ruta es un archivo	os.path.isfile(path)
Saber si una ruta es un directorio	os.path.isdir(path)
Saber si una ruta es un enlace simbólico	os.path.islink(path)
Saber si una ruta es un punto de montaje	os.path.ismount(path)

Módulo sys

- El módulo sys es el encargado de proveer variables y funcionalidades, directamente relacionadas con el intérprete.

Variable	Descripción
sys.argv	Retorna una lista con todos los argumentos pasados por línea de comandos. Al ejecutar python modulo.py arg1 arg2, retornará una lista: ['modulo.py', 'arg1', 'arg2']
sys.executable	Retorna el path absoluto del binario ejecutable del intérprete de Python
sys.maxint	Retorna el número positivo entero mayor, soportado por Python
sys.platform	Retorna la plataforma sobre la cuál se está ejecutando el intérprete
sys.version	Retorna el número de versión de Python con información adicional

Métodos del módulo sys

Método	Descripción
sys.exit()	Forzar la salida del intérprete
sys.getdefaultencoding()	Retorna la codificación de caracteres por defecto
sys.getfilesystemencoding()	Retorna la codificación de caracteres que se utiliza para convertir los nombres de archivos unicode en nombres de archivos del sistema
sys.getsizeof(object[, default])	Retorna el tamaño del objeto pasado como parámetro. El segundo argumento (opcional) es retornado cuando el objeto no devuelve nada.

Módulo random

Método	Descripción
random.randint(a, b)	Retorna un número aleatorio entero entre a y b
random.choice(secuencia)	Retorna cualquier dato aleatorio de secuencia
random.shuffle(secuencia)	Retorna una mezcla de los elementos de una secuencia
random.sample(secuencia, n)	Retorna n elementos aleatorios de secuencia

Más módulos

- Consultar la lista completa en el documento:
  - PythonStandardModules.pdf
- https://docs.python.org/3/library/index.html

4 Algunas librerías útiles

Módulos útiles, paquetes y bibliotecas

- Fuente:
- https://wiki.python.org/moin/UsefulModules

Foreign Function Interface

- **CTypes** - Un paquete para llamar a las funciones de dlls / bibliotecas compartidas.
  - Ahora se incluye con Python 2.5 y superior.
  - http://starship.python.net/crew/theller/ctypes/tutorial.html
- **Cython** es un lenguaje de extensión para el tiempo de ejecución CPython.
  - Traduce el código de Python al código C rápido y apoya la llamada externa C y código C ++ de forma nativa. A diferencia de ctypes, requiere un compilador C para traducir el código generado.
  - http://cython.org/

C Foreign Function Interface

- Interactua con casi cualquier código C de Python, basado en declaraciones similares a C que puedes copiar y pegar desde archivos de encabezado o documentación.
  - https://cffi.readthedocs.io/en/latest/
- Soporta dos modos:
  - un modo de compatibilidad **ABI** en que permite cargar dinámicamente y ejecutar funciones desde módulos ejecutables (esencialmente exponiendo la misma funcionalidad que LoadLibrary o dlopen)
  - un modo **API**, que le permite construir C.

Foreign Function Interface

```
from cffi import FFI
ffi = FFI()
ffi.cdef("size_t strlen(const char*);")
clib = ffi.dlopen(None)
length = clib.strlen("String to be evaluated.")
# prints: 23
print("{}".format(length))
```

Java

- **JCC** - es un generador de contenedor para la API Java, exponiéndolos a C ++ y Python, y produciendo extensiones de Python que se comunican a través de JNI con una máquina virtual Java.
  - https://pypi.python.org/pypi/JCC
- **Jepp** – embebe CPython en Java. Es seguro de usar en un ambiente fuertemente roscado, es bastante rápido y su estabilidad es una característica principal y objetivo.
  - https://github.com/ninia/jep
- **JPype** - permite que los programas de python tengan acceso completo a las bibliotecas de clases de java. Esto se logra a través de la interfaz a nivel nativo en ambas máquinas virtuales.
  - http://jpype.sourceforge.net/
- **Jython** - (anteriormente: JPython) es un compilador de bytecode de Python a Java. Está escrito en Java. La mayoría de los scripts de Python deben ejecutarse con poca o ninguna modificación en Jython. La excepción son los scripts que usan extensiones de Python escritas en C. Algunos módulos en la biblioteca estándar pueden no estar disponibles en Jython.
  - http://www.jython.org/Project/index.html

Networking

- **asyncoro** - framework de programación concurrente asíncrono. Posee corutinas con interfaz tipo thread
  - http://asyncoro.sourceforge.net/
- **Gevent** - Biblioteca de red basada en corutinas
  - http://gevent.org/
- **TwistedMatrix** – Framework para networking basado en eventos
  - http://twistedmatrix.com/
- **RPyC** - RPC transparente / framework de computación distribuida
  - http://rpyc.wikispaces.com/
- **PyRO** - potente OO RPC
  - http://pyro.sourceforge.net/
- **HTTPLib2** - Una biblioteca cliente HTTP bastanete completa que admite muchas características dejadas fuera de otras bibliotecas HTTP, como httplib en la biblioteca estándar.
  - http://code.google.com/p/httplib2/
- **Celery–** Cola de tareas distribuidas para procesamiento fuera de banda / RPC y más.
  - http://celeryproject.org/

HTML Forms

- **ClientForm** - "ClientForm es un módulo de Python para manejar formularios HTML en el lado del cliente, útil para analizar los formularios HTML, rellenándolos y devolviendo los formularios completados al servidor.Desarrollado desde un puerto del módulo Perl de Gisle Aas HTML :: Form , de la biblioteca libwww-perl, pero la interfaz no es lo mismo ". - desde el sitio web.
  - http://wwwsearch.sourceforge.net/ClientForm/
- **FormEncode** - es un paquete de validación y generación de formularios
  - http://www.formencode.org/en/latest/
- **lxml.html** - tiene soporte para tratar formularios en documentos HTML
  - http://lxml.de/

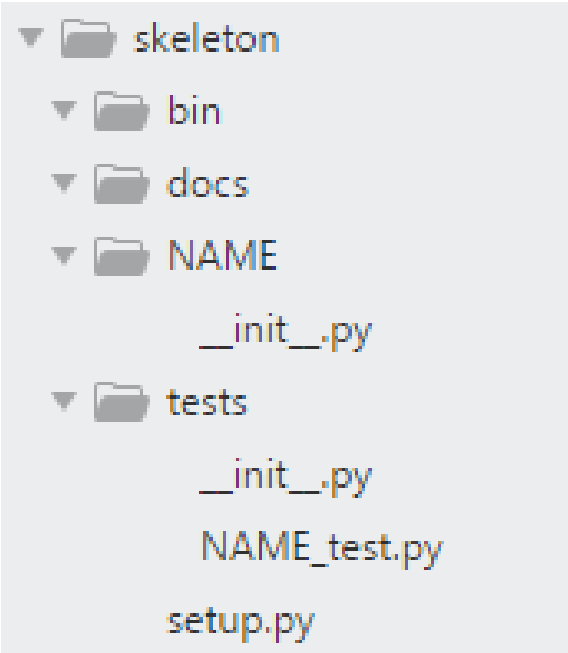
XML Processing

- **ElementTree** - El tipo de elemento es un objeto contenedor simple pero flexible, diseñado para almacenar estructuras de datos jerárquicas, como conjuntos de información XML simplificados, en la memoria.
  - Python 2.5 y superior tiene ElementTree en la Biblioteca Estándar
  - http://effbot.org/zone/element-index.htm
- **lxml** es una biblioteca muy rápida, fácil de usar y versátil para el manejo de XML que es en su mayoría compatible con pero mucho más rico en funciones que ElementTree
  - http://lxml.de/
- **Amara** - Amara proporciona herramientas en las que puede confiar para conformarse con los estándares XML sin perder la familiar sensación de Python.
  - http://wiki.xml3k.org/Amara2
- **PythonXml** proporciona una lista de soluciones de procesamiento XML disponibles.
  - https://wiki.python.org/moin/PythonXml

5 Estructura de proyectos

Esqueleto de proyecto

- Este directorio esqueleto contendrá todos los elementos básicos que se necesita para poner en marcha un proyecto nuevo: layout, pruebas automatizadas, módulos y scripts de instalación.
- Cuando se vaya a crear un nuevo proyecto, simplemente se copiará este directorio a un nuevo nombre y edite los archivos para empezar.



Esqueleto de proyecto: setup.py

- El script de configuración es el centro de toda la actividad en la construcción, distribución e instalación de módulos utilizando el Distutils
  - https://docs.python.org/3/distutils/setupscript.html
- Contiene la definición de lo necesario para el módulo
- Para ver las opciones

  python setup.py –help
- Para instalar las dependencias

  python setup.py install

Esqueleto de proyecto: setup.py

```
try:
    from setuptools import setup
except ImportError:
    from distutils.core import setup

config = {
    'description': 'My Project',
    'author': 'Ricardo A',
    'url': 'URL to get it at.',
    'download_url': 'Where to download it.',
    'author_email': 'ricardo@enmotionvalue.com',
    'version': '0.1',
    'install_requires': ['nose'],
    'packages': ['NAME'],
    'scripts': [],
    'name': 'projectname'
}

setup(**config)
```

PIP

- La herramienta recomendada por PyPA para instalar paquetes de Python.
  - PYPA: https://packaging.python.org/guides/tool-recommendations/
  - PIP: https://pip.pypa.io/en/stable/
- Para instalar cualquier módulo o paquete publicado usaremos

  pip install <modulo>
  - https://packaging.python.org/tutorials/installing-packages/
- Dentro de un proyecto podemos usar "pip install ." para instalar el paquete

Creando un proyecto

- Crea el esqueleto descrito anteriormente
- Haz una copia para un proyecto de nombre “calculador_areas”
- Instala las dependencias

6 Testing de módulos

Tests automáticos

- Python permite definir test unitarios para probar las funcionalidades de un módulo
- El objetivo de los tests es aligerar el tiempo necesario para testar el software, garantizar la regresividad del mismo y ayudar a identificar posibles errores
- Python posee dos módulos principales para testing:
  - unittest
    - https://docs.python.org/3/library/unittest.html
  - nose: hereda y complementa unittest
    - http://nose.readthedocs.io/en/latest/
    - Es necesario instalarlo usando pip: pip install nose

Escribiendo un TestCase

- Para generar un test case importaremos las tools de nose y el módulo a probar
- A continuación generaremos tantas funciones como pruebas querramos realizar
  - Las funciones deben comenzar por test
  - Dentro Realizaremos tantas aserciones como sea necesario para realizar las comprobaciones necesarias
  - Lista de métodos assert: https://docs.python.org/3/library/unittest.html#assert-methods
- Guardaremos el test en el directorio tests del proyecto
- Finalemente lanzaremos (en raíz) el comando nose para buscar y ejecutar los tests de un proyecto:

```
nosetests
```

Test Case

```
# class Room
class Room(object):

    def __init__(self, name, description):
        self.name = name
        self.description = description
        self.paths = {}

    def go(self, direction):
        return self.paths.get(direction, None)

    def add_paths(self, paths):
        self.paths.update(paths)
```

```
# test para Room
from nose.tools import *
from ex47.game import Room


def test_room():
    gold = Room("GoldRoom",
                """ This room has gold in it you
can grab. There's a \
    door to the north."""
                )
    assert_equal(gold.name, "GoldRoom")
    assert_equal(gold.paths, {})
```

Generando un test unitario

- Genera los tests unitarios para tu proyecto calculador_áreas
- Ejecuta los tests

netmind

WeKnowIT

Barcelona

C. Almogàvers, 123
08018 Barcelona
Tel. 93 304.17.20
Fax. 93 304.17.22

Madrid

Plaza Carlos Trías Bertrán, 7
28020 Madrid
Tel. 91 442.77.03
Fax. 91 442.77.07

www.netmind.es