T-IMPORTACION

Contenido

1. Definición paquete y módulo

En Python, cada uno de nuestros archivos .py se denominan módulos. Estos módulos, a la vez, pueden formar parte de paquetes.

Un paquete, es una carpeta que contiene archivos .py. Pero, para que una carpeta pueda ser considerada un paquete, debe contener un archivo de inicio llamado __init__.py. Este archivo, no necesita contener ninguna instrucción. De hecho, puede estar completamente vacío.

```
└── paquete
    ├── __init__.py
    ├── modulo1.py
    ├── modulo2.py
    └── modulo3.py
```

Los paquetes, a la vez, también pueden contener otros sub-paquetes:

```
.
└── paquete
    ├── __init__.py
    ├── modulo1.py
    └── subpaquete
        ├── __init__.py
        ├── modulo1.py
        └── modulo2.py
```

Y los módulos, no necesariamente, deben pertenecer a un paquete:

```
.
├── modulo1.py
```

```
└── paquete
    ├── __init__.py
    ├── modulo1.py
    └── subpaquete
        ├── __init__.py
        ├── modulo1.py
        └── modulo2.py
```

2. Importando módulos enteros

El contenido de cada módulo, podrá ser utilizado a la vez, por otros módulos. Para ello, es necesario importar los módulos que se quieran utilizar.

Para importar un módulo, se utiliza la instrucción `import`, seguida del nombre del paquete (si aplica) más el nombre del módulo (sin el .py) que se desee importar.

En el caso de que el modulo se encuentre dentro del mismo paquete:

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

import modulo          # importar un módulo que no pertenece a un paquete
import paquete.modulo1 # importar un módulo que está dentro de un paquete
import paquete.subpaquete.modulo1
```

La instrucción `import` seguida de nombre_del_paquete.nombre_del_modulo, nos permitirá hacer uso de todo el código que dicho módulo contenga.

Nota: Python tiene sus propios módulos, los cuales forman parte de su librería de módulos estándar, que también pueden ser importados.

3. Namespaces

Para acceder (desde el módulo donde se realizó la importación), a cualquier elemento del módulo importado, se realiza mediante el namespace, seguido de un punto (.) y el nombre del elemento que se desee obtener.

En Python, un namespace, es el nombre que se ha indicado luego de la palabra `import`, es decir la ruta (namespace) del módulo:

```
print modulo.CONSTANTE_1
print paquete.modulo1.CONSTANTE_1
print paquete.subpaquete.modulo1.CONSTANTE_1
```

4. Alias

Es posible también, abreviar los namespaces mediante un alias. Para ello, durante la importación, se asigna la palabra clave as seguida del alias con el cual nos referiremos en el futuro a ese namespace importado:

```
import modulo as m
import paquete.modulo1 as pm
import paquete.subpaquete.modulo1 as psm
```

Luego, para acceder a cualquier elemento de los módulos importados, el namespace utilizado será el alias indicado durante la importación:

```
print m.CONSTANTE _1
print pm.CONSTANTE _1
print psm.CONSTANTE_1
```

5. Importar módulos sin utilizar namespaces

En Python, es posible también, importar de un módulo solo los elementos que se desee utilizar. Para ello se utiliza la instrucción from seguida del namespace, más la instrucción import seguida del elemento que se desee importar:

```
from paquete.modulo1 import CONSTANTE_1
```

En este caso, se accederá directamente al elemento, sin recurrir a su namespace:

```
print CONSTANTE_1
```

Es posible también, importar más de un elemento en la misma instrucción. Para ello, cada elemento irá separado por una coma (,) y un espacio en blanco:

```
from paquete.modulo1 import CONSTANTE_1, CONSTANTE_2
```

Pero ¿qué sucede si los elementos importados desde módulos diferentes tienen los mismos nombres? En estos casos, habrá que **prevenir fallos, utilizando alias para los elementos:**

```
from paquete.modulo1 import CONSTANTE_1 as C1, CONSTANTE_2 as C2
from paquete.subpaquete.modulo1 import CONSTANTE_1 as CS1, CONSTANTE_2 as CS2

print C1
print C2
print CS1
print CS2
```

6. Orden de importación

La importación de módulos debe realizarse al comienzo del documento, en orden alfabético de paquetes y módulos. (aunque hacerlo en otro orden y a lo largo del documento no genera errores)

- Primero deben importarse los módulos propios de Python.
- Luego, los módulos de terceros
- y finalmente, los módulos propios de la aplicación.

Entre cada bloque de imports, debe dejarse una línea en blanco.

De forma alternativa (pero muy poco recomendada), también es posible importar todos los elementos de un módulo, sin utilizar su namespace pero tampoco alias. Es decir, que todos los elementos importados se accederán con su nombre original:

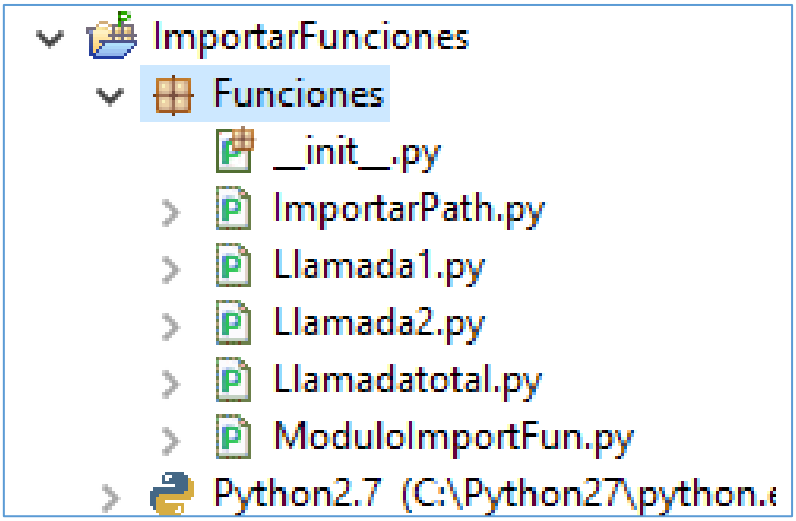
```
from paquete.modulo1 import *

print CONSTANTE_1
print CONSTANTE_2
```

7. Importar módulos de otros proyectos

Hay que incluirlos como librerías externas y hacer una importación normal

8. Ejemplos:

Dado este proyecto con los siguientes módulos dentro de un mismo paquete



- Modulo ImportFun.py:

Este módulo no tiene la sentencia inicial `if __name__ == '__main__':`

Si la tuviera ninguna de las funciones de su interior podrían ser llamadas ya que indica que el código que sigue se ejecute únicamente cuando es programa principal.

```
#-*- coding: utf-8 -*-
'''
Created on 01/12/2015

@author: Amaia
'''


def basica():
    print "Sin argumentos ni return"
    print "Llamada a basica"
    print "retorno:",basica()

def basica_return():
    print "Sin argumentos y return"
    return "Hola"

def basica_argumentos(arg1,arg2):
    print "Con argumentos y return"
    arg1=arg1+arg2
    return arg1

def argumentos (arg1,arg2,arg3=23,arg4=(4,True)):

    print arg1,arg2,arg3,arg4
```

- Módulo Llamada1:
  Este módulo importa el módulo anterior referenciándolo por el nombre del paquete. Módulo. Esto indica que a partir de ahí, cuando se quiera utilizar una función de ese módulo, deberá hacerse con ese prefijo: paquete. Módulo

```
#-*- coding: utf-8 -*-
'''
Created on 07/12/2015

@author: Amaia
'''
#Llamada utilizando namespace = paquete.Modulo
import Funciones.ModuloImportFun

if __name__ == '__main__':

    print Funciones.ModuloImportFun.basica_return()
```

- Módulo Llamada2:

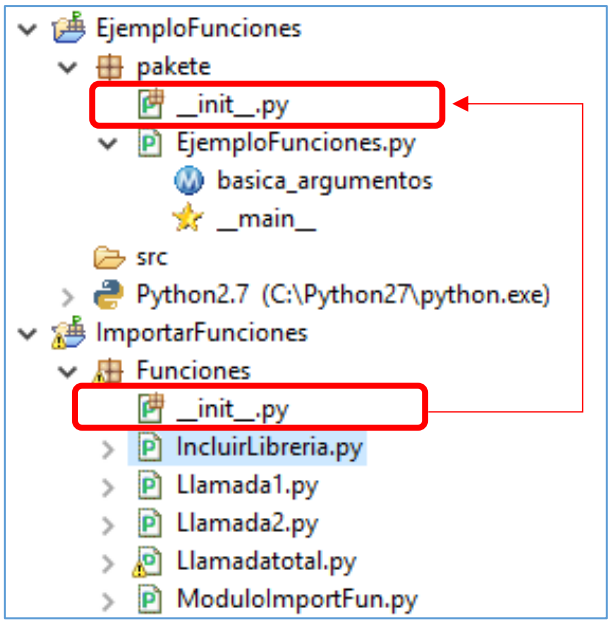
Este módulo importa el módulo sólo con el nombre. Esto es posible porque están dentro del mismo paquete. Para poder utilizar las funciones que están dentro de él, utilizará como prefijo solamente el nombre del Módulo que se importa.

```
#-*- coding: utf-8 -*-
'''
Created on 07/12/2015

@author: Amaia
'''
#Llamada utilizando namespace = Modulo
import ModuloImportFun

if __name__ == '__main__':

    print ModuloImportFun.basica_return()
```

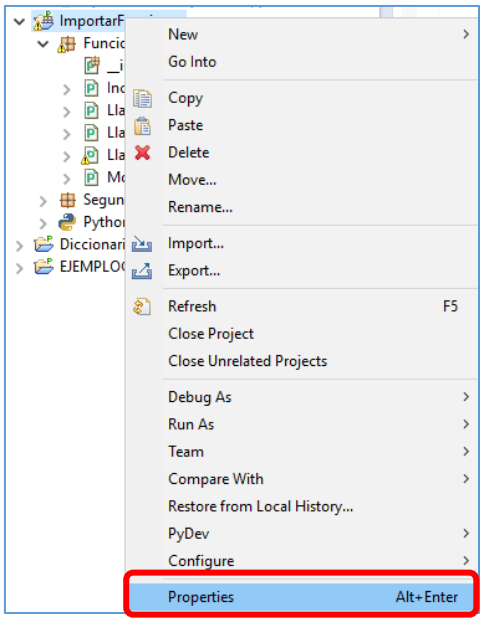
- Módulo Llamada Total:
  Este módulo importa desde el módulo ModuloImportFun solamente la función básica_return. Para nombrarla no se utilizará como prefijo el namespace, eso sí únicamente podrá utilizarse esa función del módulo nombrado. Si se quisieran importar más elementos (funciones, variables,…) se pondrían en l misma línea separadas por coma. (`from ModuloImportFun import basica_return, basica_argumentos`)

```
'''
Created on 07/12/2015

@author: Amaia
'''
from ModuloImportFun import basica_return
#Para importar todos los elementos del modulo: from ModuloImportFun
import *
if __name__ == '__main__':

    print basica_return()
```

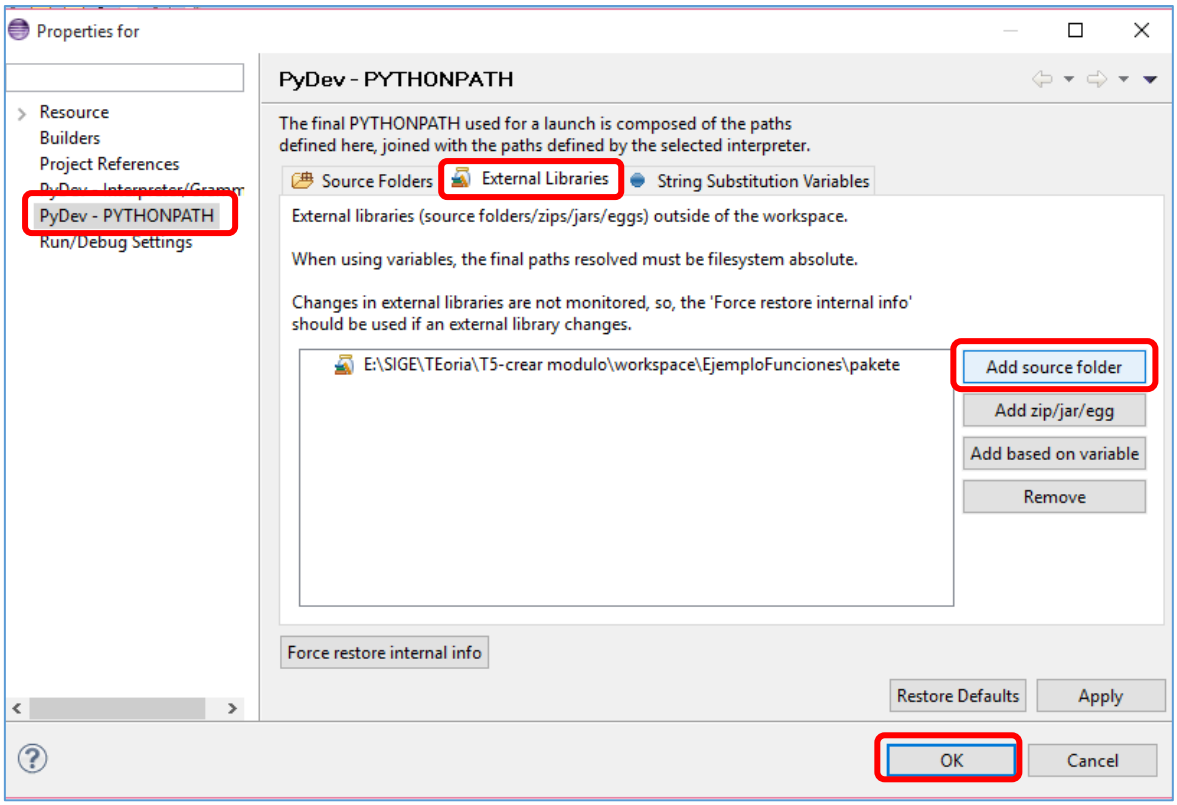
- Módulo Incluir Librería:
  Si queremos utilizar el código que está en un paquete que ni siquiera pertenece al proycto en el que estamos, debemos incluirla como Librería externa. Por ejemplo en este caso, desde uno de los módulos queremos utilizar código que se encuentra en el Proyecto EjemploFunciones/paquete/EjemploFunciones.

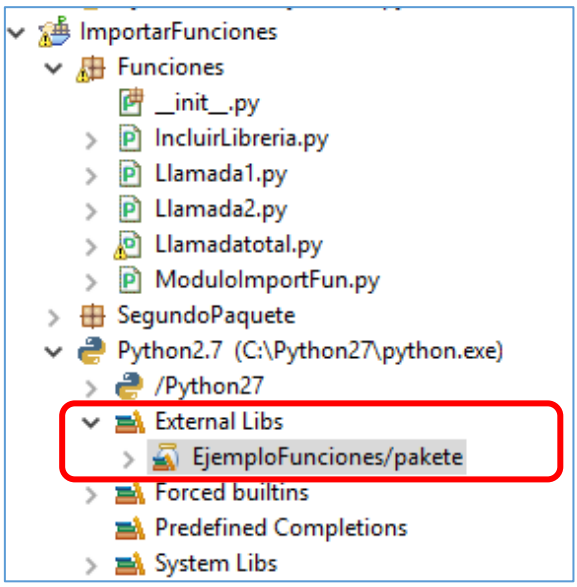


Desde el proyecto actual→ Properties



PyDEv-PYTHONPATH → External Libraies→ Add source folder



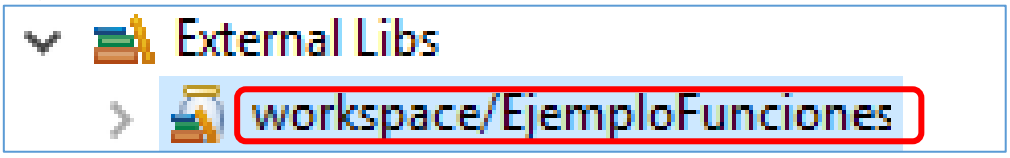


En este caso la librería externa se ha referenciado a nivel de paquete. A la hora de importar, puede hacerse directamente por el nombre del módulo sin ser necesario referenciar el paquete. Además de ha utilizado un alias:

```
'''
Created on 07/12/2015

@author: Amaia
'''
import EjemploFunciones as EF

if __name__ == '__main__':
```

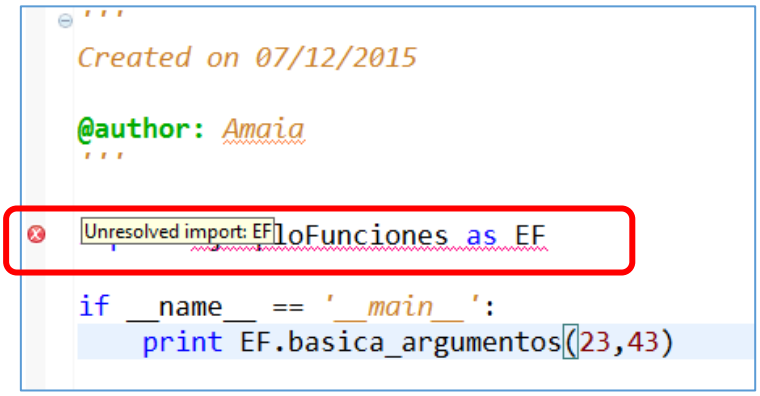
```
print EF.basica_argumentos(23,43)
```

Si la librería externa se hubiera referenciado solamente a nivel del proyecto (sin el nombre del paquete):



El código daría error si no se incluye en el namespace el nombre del paquete:

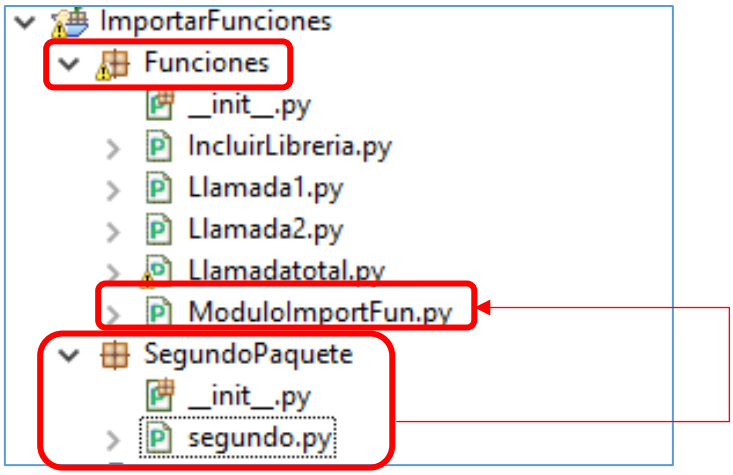
```
'''
Created on 07/12/2015

@author: Amaia
'''
import EjemploFunciones as EF
if __name__ == '__main__':
    print EF.basica_argumentos(23,43)
```



Habría que hacer el import indicando el paquete:

```
'''
Created on 07/12/2015

@author: Amaia
'''
import pakete.EjemploFunciones as EF
if __name__ == '__main__':
    print EF.basica_argumentos(23,43)
```

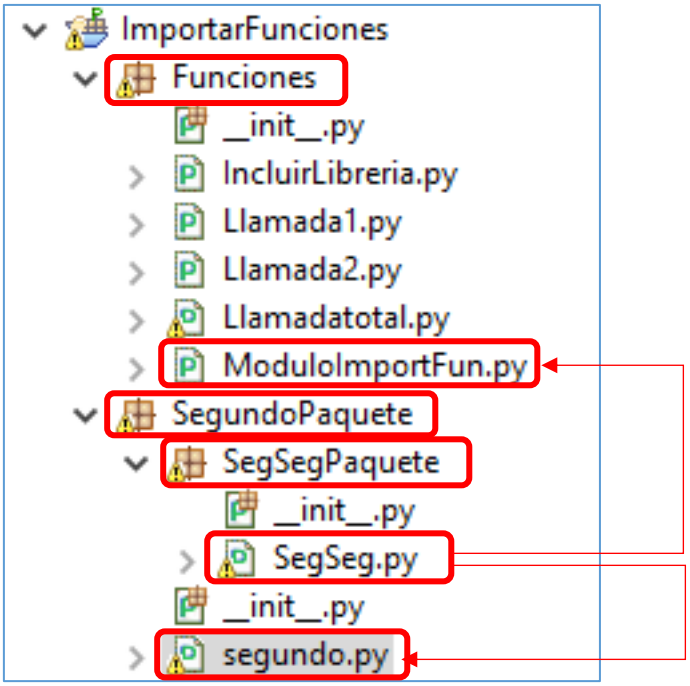
- Llamada al código de un módulo que está en otro paquete
  Ejemplo1:



Módulo segundo.py:

```
'''
Created on 08/12/2015

@author: Amaia
'''
import Funciones.ModuloImportFun as FM
if __name__ == '__main__':
          FM.basica()
```

Ejemplo2:



Código SegSeg.py:

```
'''
Created on 08/12/2015

@author: Amaia
'''
import Funciones.ModuloImportFun #No tiene en cuenta el
nivel. Lo busca dentro del directorio del proyecto.
```

```
import SegundoPaquete.segundo as Seg

if __name__ == '__main__':
        Funciones.ModuloImportFun.basica()
        Seg.FM.basica() #Desde el propio Seg, se puede
hacer referencia al alias hecho en Seg (FM) y utilizar las
funciones de ese módulo importado
```

Más información: https://www.python.org/dev/peps/pep-0008/#imports

9. Biblioteca estándar

1. Interfaz al sistema operativo (**os**)

   El módulo os provee docenas de funciones para interactuar con el sistema operativo.

```
import os

if __name__ == '__main__':
    os.getcwd()      # devuelve el directorio de trabajo actual
    os.chdir('./ahora')   # cambia el directorio de trabajo
    os.system('mkdir ahora')   # ejecuta el comando 'mkdir' en el
sistema
    dir(os)  #devuelve una lista de todas las funciones del modulo
    help(os) # devuelve un manual creado a partir de las
documentaciones del modulo.

    os.system('rmdir ahora')   # ejecuta el comando 'rmdir' en el
sistema
```

   Para tareas diarias de administración de archivos y directorios, el módulo `shutil` provee una interfaz de más alto nivel que es más fácil de usar:

```
import shutil
shutil.copyfile('datos.db', 'archivo.db')
shutil.move('./ahora/ejecutables', 'dir_instalac')
```

2. Comodines de archivos

El módulo `glob` provee una función para hacer listas de archivos a partir de búsquedas con comodines en directorios.

```
import glob
print glob.glob('*.py')
```

3. Argumentos de línea de órdenes

Los programas frecuentemente necesitan procesar argumentos de línea de órdenes. Estos argumentos se almacenan en el atributo *argv* del módulo `sys` como una lista. argv[0] es el nombre de lo que se ejecuta:

```
import sys
print(sys.argv)
```

4. Redirección de la salida de error y finalización del programa
   El módulo `sys` también tiene atributos para *stdin*, *stdout*, y *stderr*. Este último es útil para emitir mensajes de alerta y error para que se vean incluso cuando se haya redireccionado *stdout*:

```
import sys
sys.stderr.write('Alerta, archivo de log no encontrado\n')
#La forma más directa de terminar un programa es usar sys.exit()
```

5. Matemática

El módulo math permite el acceso a las funciones de la biblioteca C subyacente para la matemática de punto flotante:

```
import math
print math.cos(math.pi / 4)
print math.log(1024, 2)
```

El módulo `random` provee herramientas para realizar selecciones al azar:

```
import random
    print random.choice(['manzana', 'pera', 'banana'])
          print random.sample(range(100), 10) #10 numeros
enteros al azar del 0-99
    print random.random()    # un float al azar
    print random.randrange(6)    # un entero al azar de 0-5
```

Función/ variable	Explicación	Equivalente
pi	Número pi	
e	Número e	
trunc	Truncar un número	
ceil	Redondeo superior	
floor	Redondeo inferior	
pow	Potencia (dos argumentos)	
sqrt	Raíz cuadrada	X**0.5
degrees	Permite convertir radianes en grados	X*180/Pi
radians	Conversión inversa	X*Pi/180
cos	Coseno	
sin	Seno	
tan	Tangente	
acos	Función recíproca de coseno	
asin	Función recíproca de seno	
atan	Función recíproca de tangente	

atan2	Tangente inversa de un punto del plano X, Y	tan(x/Y)
cosh	Coseno hiperbólico	
sinh	Seno hiperbólico	
tanh	Tangente hiperbólica	
acosh	Función recíproca de coseno hiperbólico	
asinh	Función recíproca de seno hiperbólico	
atanh	Función recíproca de tangente hiperbólica	
hypoth	Llamada hipotenusa debido al triángulo rectángulo formado por el eje de abscisas y la recta paralela a la de ordenadas que pasa por el punto X, Y. Se trata de la norma, es decir, la distancia entre el punto y el origen	(x**2+y**2)**0.5
exp	Función exponencial	e**X
expm1	Utilizada para pequeños números reales, permite obtener una mejor precisión	e**X-1
log	Función logaritmo neperiano, puede recibir la base como segundo argumento	
log1p	Logaritmo de 1+X. Permite obtener una mejor precisión para los números próximos a cero	log(1+X)
log10p	Función logaritmo en base 10. Más preciso que log(X, 10)	log(X) / log(10)
frexp	Devuelve la mantisa y el exponente	
ldexp	Función inversa de la anterior, posee dos argumentos M y E	M * 2**E
fmod	Función módulo para números reales (flotantes) con mejor precisión que el operador módulo (%)	X%Y, aunque con mejor precisión
modf	Devuelve la parte tras la coma y la parte entera de una cifra, siendo ambas números reales	
fabs	Valor absoluto	abs
fsum	Suma (mejor precisión, véase fmod)	sum

6. Fechas y tiempos

El módulo `datetime` ofrece clases para manejar fechas y tiempos tanto de manera simple como compleja.

Aunque soporta aritmética sobre fechas y tiempos, el foco de la implementación es en la extracción eficiente de partes para manejarlas o formatear la salida. El módulo también soporta objetos que son conscientes de la zona horaria.

```
from datetime import date
hoy = date.today()
print hoy
print hoy.strftime("%m-%d-%y. %d %b %Y es %A. hoy es %d de %B.")

nacimiento = date(1964, 7, 31)
edad = hoy - nacimiento
print edad.days
```

7. Compresión de datos

Los formatos para archivar y comprimir datos se soportan directamente con los módulos: `zlib`, `gzip`, `bz2`, `lzma`, `zipfile` y `tarfile`.

```
import zlib
s = b'No me preocupa tanto la gente mala, sino el espantoso
silencio de la gente buena'
print len(s)
t = zlib.compress(s)
#print t
print len(t)
print zlib.decompress(t)
print zlib.crc32(s)
```

Métodos disponibles en zlib:

```
'adler32', 'compress', 'compressobj', 'crc32', 'decompress',
'decompressobj', 'error'
```

Más información:

- Operaciones fechas y horas: http://python-para-impacientes.blogspot.com.es/2014_02_01_archive.html
- docs.python.org.ar/tutorial/3/stdlib.html
- http://codehero.co/python-desde-cero-modulos/