TEMA III

DESARROLLO DE APLICACIONES DE PROPÓSITO GENERAL

"Un sueño no se hace realidad por arte de magia, necesita sudor, determinación y trabajo duro"
Colin Powell

Módulos y Paquetes

Módulo: Cada uno de los ficheros py que nosotros creamos se llama módulo. Los elementos creados en un módulo (funciones, clases, …) se pueden importar para ser utilizados en otro módulo. Python permite importar el módulo completo o una parte. El nombre que vamos a utilizar para importar un módulo es el nombre del fichero.

Para importar un módulo completo tenemos que utilizar las instrucción import

import M : Ejecuta el código que contiene M y crea una referencia a él, de manera que pueden invocarse un objeto o función f definida en él mediante la sintaxis M.f.

import M as N : Ejecuta el código que contiene M y crea una referencia a él con el nombre N, de manera que pueden invocarse un objeto o función f definida en él mediante la sintaxis N.f. Esta forma es similar a la anterior, pero se suele usar cuando el nombre del módulo es muy largo para utilizar un alias más corto.

Ejemplo: Aplica este concepto a los dos ejercicios de funciones de la práctica$_3$

No tenemos por qué importar un módulo entero, podemos importar aquellas funciones o variables o clases que nos interesen con **from import**

from M import f, g, ... : Ejecuta el código que contiene M y crea referencias a los objetos f, g, ..., de manera que pueden ser invocados por su nombre. De esta manera para invocar cualquiera de estos objetos no hace falta precederlos por el nombre del módulo, basta con escribir su nombre.

```
>>> from potencias import cubo
>>> cubo(3)
27
```

Podemos importar varios elementos separándolos con comas:

```
>>> from potencias import cubo,cuadrado
```

Podemos tener un problema al importar dos elementos de dos módulos que se llamen igual. En este caso tengo que utilizar alias:

```
>>> from potencias import cuadrado as pc
>>> from dibujos import cuadrado as dc
>>> pc(3)
9
>>> dc()
```

from M import * : Ejecuta el código que contiene M y crea referencias a todos los objetos públicos (aquellos que no empiezan por el carácter _) definidos en el módulo, de manera que pueden ser invocados por su nombre.

```
>>> from math import *
>>> cos(pi)
-1.0
```

Módulos de la librería estándar más importantes

Python viene con una biblioteca de módulos predefinidos que no necesitan instalarse. Están escritos en lenguaje CAlgunos de los más utilizados son:

- sys: Funciones y parámetros específicos del sistema operativo.
- os: Interfaz con el sistema operativo.
- os.path: Funciones de acceso a las rutas del sistema.
- io: Funciones para manejo de flujos de datos y ficheros.
- string: Funciones con cadenas de caracteres.
- datetime: Funciones para fechas y tiempos.
- math: Funciones y constantes matemáticas.
- statistics: Funciones estadísticas.
- random: Generación de números pseudo-aleatorios.

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.

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)`

Descripción	Método
Crear un enlace simbólico	`os.symlink(path, nombre_destino)`

```
>>> import os
>>> os.getcwd()
'/home/jose/github/curso_python3/curso/u40'
>>> os.chdir("..")
>>> os.getcwd()
'/home/jose/github/curso_python3/curso'
```

El módulo os también nos provee del submódulo path (os.path) el cual nos 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)`

Descripción	Método
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)`

Tenemos módulos para trabajar con fechas y horas

El módulo Time trabaja las fechas tal y como lo hace C, segundos transcurridos desde el 1 del 1 de 1970. Tampoco podemos manejar fechas más allá de 2038 porque se produce un desbordamiento en el tipo float.

Disponemos de los módulos datetime y calendar para trabajar de una formas "mas amigable", amplían las posibilidades del módulo time que provee funciones para manipular expresiones de tiempo.

El módulo datetime contiene las clases datetime,date,time y timedelta. Esta última representa una "duración"(la diferencia entre dos horas o fechas)

```
>>> from datetime import datetime //Estamos importando la clase datetime del módulo
>>> datetime.now()
>>> datetime.now().day, datetime.now().month, datetime.now().year
(4, 3, 2017)
```

Para compara fechas y horas:

```
>>> from datetime import datetime, date, time, timedelta
>>> hora1 = time(10,5,0)
>>> hora2 = time(23,15,0)
>>> hora1>hora2
False
```

```
>>> fecha1=date.today()
>>> fecha2=fecha1+timedelta(days=2)
>>> fecha1<fecha2
True
```

Podemos imprimir aplicando un formato:

```
>>> fecha1.strftime("%d/%m/%Y")
'04/03/2017'
>>> hora1.strftime("%H:%M:%S")
'10:05:00'
```

Podemos convertir una cadena a un datetime:

```
>>> tiempo = datetime.strptime("12/10/2017","%d/%m/%Y")
```

Y podemos trabajar con cantidades (segundos, minutos, horas, días, semanas,...) con timedelta:

```
>>> hoy = date.today()
>>> ayer = hoy – timedelta(days=1)
>>> diferencia=hoy -ayer
>>> diferencia
datetime.timedelta(1) //retornará 1 day 00:00:00
```

```
>>> fecha1=datetime.now()
>>> fecha2=datetime(1995,10,12,12,23,33)
>>> diferencia=fecha1-fecha2
>>> diferencia
datetime.timedelta(7813, 81981, 333199)
```

En cuanto al módulo calendar

Podemos obtener el calendario del mes actual:

```
>>> año = date.today().year
>>> mes = date.today().month
>>> calendario_mes = calendar.month(año, mes)
>>> print(calendario_mes)
    March 2017
Mo Tu We Th Fr Sa Su
      1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
```

Y para mostrar todos los meses del año:

```
>>> print(calendar.TextCalendar(calendar.MONDAY).formatyear(2017,2, 1, 1, 2))
```

Con la función dir podemos obtener los elementos definidos en un módulo

Otras librerías

Estas librerías no vienen en la distribución estándar de Python y necesitan instalarse.

- NumPy: Funciones matemáticas avanzadas y arrays.
- SciPy: Más funciones matemáticas para aplicaciones científicas.
- matplotlib: Análisis y representación gráfica de datos.
- Pandas: Funciones para el manejo y análisis de estructuras de datos.
- Request: Acceso a internet por http.

Paquete: Para estructurar nuestros módulos podemos crear paquetes, Un paquete es una carpeta que contiene archivos .py que serán módulos. 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. Los paquetes, a la vez, también pueden contener otros sub-paquetes.

Para importar un módulo del paquete import paquete.modulo

Si tenemos subpaquetes import paquete.subpaquete.modulo

```
>>> import operaciones.potencias
>>> operaciones.potencias.cubo(3)
27
```

La sintaxis from operaciones import potencia es análoga

Podemos importar un elemento de un módulo de un paquete

```
>>> from operaciones.potencias import cubo
>>> cubo(3)
```

```
Para importar varios módulos del paquete se enumeran separados por ,
    from operaciones import potencias,sumas
```

Paquetes distribuibles: empaquetar nuestros módulos para que puedan ser instalados y por tanto utilizados estén donde estén. Por defecto python busca en el directorio actual o en las rutas almacenadas en syspath. Para esto, hay que instalar nuestro paquete dentro de python.

1. Creamos un archivo setup.py que contiene una descripción de nuestro paquete: autor, nombre,...
No es necesario que este archivo esté dentro de la carpeta del paquete, lo crearemos fuera pero a su altura.
Para ello usaremos el módulo setuptools de python. Este módulo tiene una función setup que será la que utilizaremos para describir nuestro paquete

```
from setuptools import setup
setup(
    name="paqueteEjemplo",
    author="Rosa",
    version="1.0",
    description="Paquete ejemplo de operaciones y conversión horas,minutos,segundos",
    author_email="aaaa@gmail.com", #opcional
    url="wwww.salesianas.es", #opcional
    packages=["Funciones","Funciones.MasFunciones"]
)
```

2. Para generar el archivo a instalar: desde consola nos vamos al directorio donde hemos creado el archivo setup y ejecutamos el comando ***python3 setup.py sdist***

```
rosa@rosa-Essential14L:~/Escritorio/PYTHON/EjemploPaquete$ python3 setup.py sdist
running sdist
running egg_info
creating paqueteEjemplo.egg-info
writing paqueteEjemplo.egg-info/PKG-INFO
writing dependency_links to paqueteEjemplo.egg-info/dependency_links.txt
writing top-level names to paqueteEjemplo.egg-info/top_level.txt
writing manifest file 'paqueteEjemplo.egg-info/SOURCES.txt'
reading manifest file 'paqueteEjemplo.egg-info/SOURCES.txt'
writing manifest file 'paqueteEjemplo.egg-info/SOURCES.txt'
warning: sdist: standard file not found: should have one of README, README.rst, README.txt, README.md

running check
creating paqueteEjemplo-1.0
creating paqueteEjemplo-1.0/Funciones
creating paqueteEjemplo-1.0/Funciones/MasFunciones
creating paqueteEjemplo-1.0/paqueteEjemplo.egg-info
copying files to paqueteEjemplo-1.0...
```

Nos crea dos carpetas info, dist. En esta última tendremos un archivo comprimido. Ese es el archivo que podemos distribuir para que sea instalados

3. Instalamos: desde conslola, en la carpeta ***dist pip3 install archivo***

```
rosa@rosa-Essential14L:~/Escritorio/PYTHON/EjemploPaquete/dist$  pip3 install paqueteEjemplo-1.0.tar.gz
Processing ./paqueteEjemplo-1.0.tar.gz
Building wheels for collected packages: paqueteEjemplo
  Building wheel for paqueteEjemplo (setup.py) ... done
  Created wheel for paqueteEjemplo: filename=paqueteEjemplo-1.0-py3-none-any.whl size=2308 sha256=6254028
951cb2a1b235a843efe3f5ae19ebe064454ce90a3baefbedc764583b9
  Stored in directory: /home/rosa/.cache/pip/wheels/bc/ee/95/2b9260256f228216ec9e86138ddd8ffbdac184ee87fe
446e3c
Successfully built paqueteEjemplo
Installing collected packages: paqueteEjemplo
Successfully installed paqueteEjemplo-1.0
rosa@rosa-Essential14L:~/Escritorio/PYTHON/EjemploPaquete/dist$
```

Ahora ya podemos usar todo el paquete independientemente de donde se encuentre el script de python desde dónde lo llamemos.

Si queremos desisntalar un paquete pip3 uninstall nombrepaquete

```
rosa@rosa-Essential14L:~/Escritorio/PYTHON/EjemploPaquete/dist$ pip3 uninstall paqueteEjemplo
Found existing installation: paqueteEjemplo 1.0
Uninstalling paqueteEjemplo-1.0:
  Would remove:
    /home/rosa/.local/lib/python3.8/site-packages/Funciones/*
    /home/rosa/.local/lib/python3.8/site-packages/paqueteEjemplo-1.0.dist-info/*
Proceed (y/n)? y
  Successfully uninstalled paqueteEjemplo-1.0
rosa@rosa-Essential14L:~/Escritorio/PYTHON/EjemploPaquete/dist$
```

Podemos comprobar ahora que a no ser que nuestro script esté en la misma ruta que el paquete, éste no lo reconocerá.