



GOBIERNO
DE ESPAÑA

MINISTERIO
DE HACIENDA
Y FUNCIÓN PÚBLICA

INAP
INSTITUTO NACIONAL DE
ADMINISTRACIÓN PÚBLICA

Introducción a la programación en Python

Tema 6. Módulos y paquetes

Autor: Francisco Mariño Ruiz

Contenido

1	Módulos y paquetes	3
1.1	Definición	3
1.2	Importando módulos.....	5
1.3	Otras consideraciones	9
1.4	Ejercicios propuestos.....	10
2	Algunos módulos propios de Python	11
2.1	Módulo time	11
2.2	Módulos de sistema	16
2.3	Otros módulos.....	18
2.4	Ejercicios propuestos.....	20
3	Repositorios.....	21
4	Soluciones a los ejercicios	25
4.1	Ejercicio 1	25
4.2	Ejercicio 2	26
5	Bibliografía.....	28

1 Módulos y paquetes

1.1 Definición

En Python, cada uno de los archivos “*.py” se denomina módulo. Estos módulos, a su vez podrán formar partes de paquetes, que son carpetas que contienen archivos “*.py”. Sin embargo, para que nuestra carpeta con módulos pueda ser considerada un paquete, además de los módulos, ha de contener un archivo de inicio llamado “__init__.py”. Este archivo puede estar vacío, pero debe existir, para que se reconozca la carpeta como paquete. Si trabajamos con un entorno de desarrollo como PyCharm, cuando creamos un paquete, el mismo se encargará de crear dicho archivo.

Para crear un paquete desde PyCharm, vamos al explorador del proyecto, y hacemos clic con el botón derecho sobre el nombre de nuestro proyecto, en el menú contextual seleccionamos “New → Python Package”.

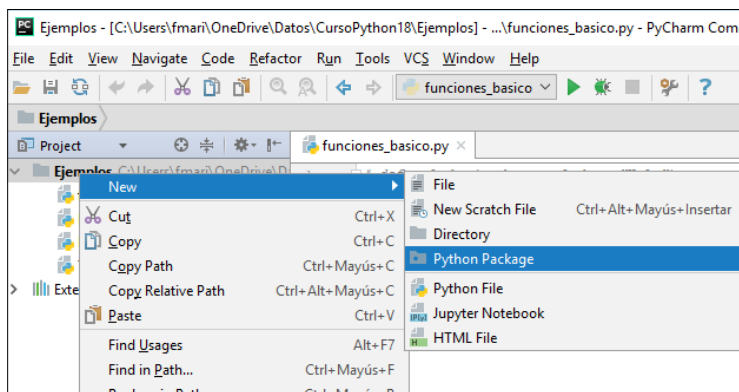


Imagen 1: Creación de nuevo paquete.

Le damos un nombre a nuestro nuevo paquete de Python:

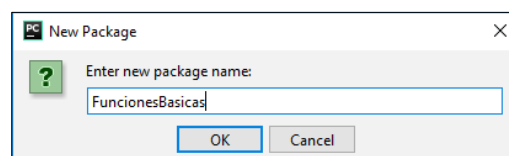


Imagen 2: Nombrar nuevo paquete.

Y aparecerá en el árbol de nuestro proyecto con el archivo “__init__.py” en su interior.

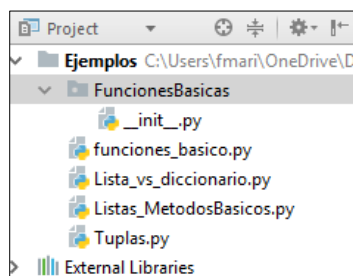


Imagen 3: Árbol con nuevo paquete.

De esta manera ya podremos añadir a nuestro paquete archivos “*.py”, que podremos usar en nuestro proyecto. Del mismo modo podemos incluso crear sub-paquetes que contengan funciones o crear módulos que no estén dentro de ningún paquete. Todo esto nos facilitará gestionar el contenido de nuestro proyecto a medida que vaya creciendo.

1.2 Importando módulos

Los módulos que creemos en nuestros paquetes, generalmente contendrán funciones que vamos a poder utilizar en cualquier script de nuestro proyecto, para lo cual, tendremos que importar los módulos. Según la ubicación del módulo tendremos que importarlo de una manera u otra.

En la siguiente imagen tenemos un ejemplo de árbol de proyecto, tenemos como programa principal el archivo “Calcular.py”, tenemos el módulo “saludar.py”, que no pertenece a ningún paquete, el paquete “FuncionesBasicas” en el que tenemos los módulos “dividir.py”, “multiplicar.py”, “restar.py” y “sumar.py” y el sub-paquete “OtrasFunciones” que contiene el módulo “modulo.py”.

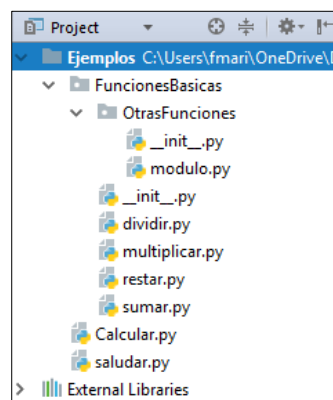
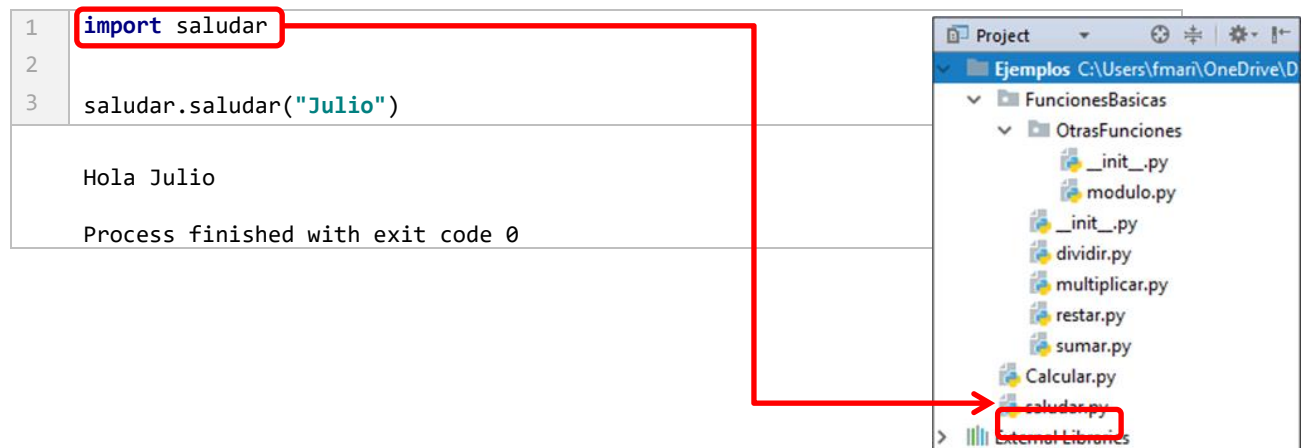


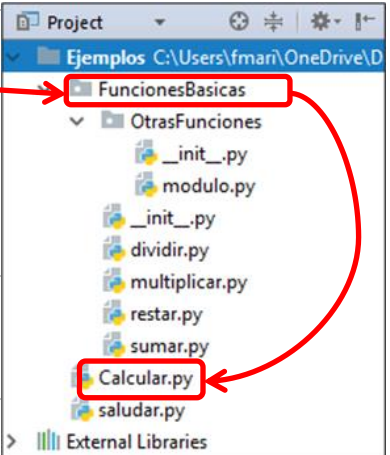
Imagen 4: Ejemplo de árbol de proyecto con múltiples paquetes.

Para poder utilizar el módulo “saludar.py” vamos a tener que importarlo en nuestro script. Al no encontrarse en ningún paquete y estar en el mismo nivel que nuestro programa principal, simplemente utilizaremos la palabra clave “import” seguida del nombre del módulo que queremos importar (sin extensión).

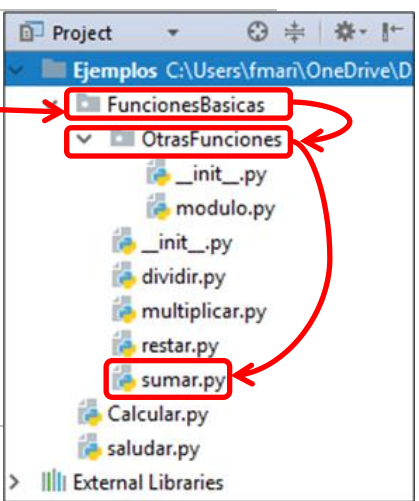
Una vez hayamos importado el módulo, podemos usar las funciones que contenga llamando al módulo, seguido de un punto y el nombre de la función que queremos utilizar con sus parámetros entre paréntesis. En este caso la función a utilizar se llama igual que el módulo que la contiene, pero no siempre tiene que ser así pues un módulo puede contener varias funciones independientes.



En segundo lugar, vamos a importar un módulo que esté dentro de un paquete, en este caso, vamos a importar el módulo “*sumar*” del paquete “*FuncionesBasicas*”. La base es semejante al caso anterior, utilizaremos la palabra clave “*import*” seguido del nombre del paquete, un punto y el nombre del módulo. Una vez importado el módulo, para utilizar las funciones que contenga habrá que invocarlo utilizando el nombre del paquete, seguido de un punto, el nombre del módulo, otro punto, el nombre de la función y paréntesis con los parámetros.

<pre>1 import saludar 2 import FuncionesBasicas.sumar 3 4 saludar.saludar("Julio") 5 6 a = FuncionesBasicas.sumar.sumar(3, 4) 7 print(a)</pre>	
<pre>Hola Julio 7 Process finished with exit code 0</pre>	

Del mismo modo, para importar una función que este dentro de un módulo que a su vez está en un sub-paquete hará que el comando de importación sea más largo obligándonos a poner todo el camino hasta el módulo “*import nombre_del_paquete.nombre_del_sub_paquete.modulo*” y para llamar a las funciones, de manera semejante a los casos anteriores.

<pre>1 import saludar 2 import FuncionesBasicas.sumar 3 import FuncionesBasicas.OtrasFunciones.modulo 4 5 saludar.saludar("Julio") 6 7 a = FuncionesBasicas.sumar.sumar(3, 4) 8 print(a) 9 10 a = FuncionesBasicas.OtrasFunciones.modulo.modulo(13, 2) 11 print(a)</pre>	
<pre>Hola Julio 7 1 Process finished with exit code 0</pre>	

Esta notación nos puede plantear un problema, ya que a medida que se añaden paquetes y subpaquetes la notación se vuelve cada vez más compleja. El lenguaje Python nos da la posibilidad de abreviar los nombres de los módulos mediante los alias. Para ello, durante la importación, se usa la palabra clave “*as*” seguida del alias con el cual nos referiremos a los módulos cuando los queramos usar.

<pre>1 import saludar as sal 2 import FuncionesBasicas.sumar as sum</pre>

```
3 import FuncionesBasicas.OtrasFunciones.modulo as mod
4
5 sal.saludar("Julio")
6
7 a = sum.sumar(3, 4)
8 print(a)
9
10 a = mod.modulo(13, 2)
11 print(a)
```

```
Hola Julio
7
1
Process finished with exit code 0
```

Otra posibilidad que nos brinda el lenguaje es importar directamente los módulos que hay dentro de un paquete, de tal manera que cuando vayamos a usar la función no tengamos que referenciar todo el esquema donde se contiene el módulo. Para ello utilizamos la palabra clave “*from*” seguida del nombre del paquete (o de la estructura de paquetes), más la palabra clave “*import*” y el nombre del paquete que deseamos importar.

```
1 import salutar
2 from FuncionesBasicas import sumar
3 from FuncionesBasicas.OtrasFunciones.modulo import modulo
4
5 salutar.saludar("Julio")
6
7 a = sumar.sumar(3, 4)
8 print(a)
9
10 a = modulo(13, 2)
11 print(a)
```

```
Hola Julio
7
1
Process finished with exit code 0
```

Utilizando este método de importación, podremos importar varios módulos o funciones a la vez separándolos por comas.

```
1 from FuncionesBasicas import sumar, restar, multiplicar, dividir
```

La combinación de *from* e *import*, también puede ser usada para importar directamente una función de un módulo.

```
1 from salutar import salutar
2 from FuncionesBasicas.sumar import sumar
3 from FuncionesBasicas.OtrasFunciones.modulo import modulo
4
5 salutar("Julio")
6
7 a = sumar(3, 4)
```

```
8 print(a)
9
10 a = modulo(13, 2)
11 print(a)
```

```
Hola Julio
7
1
Process finished with exit code 0
```

Es importante recordar que en caso de que queramos utilizar este método de importación para importar múltiples funciones, prestemos mucha atención a los nombres de las mismas, ya que si existen varias funciones de distintos módulos con el mismo nombre, podríamos incurrir en errores, por lo que en estos casos se recomienda importar las funciones combinando su importación selectiva con el uso de alias.

```
1 from aritmetica import sumar as sum_arit
2 from matrices import sumar as sum_matri
```

Si quisiésemos importar todas las funciones de un módulo utilizaríamos un asterisco y podríamos llamarlas directamente por su nombre original, pero en general, no se recomienda utilizar este método ya que hace que se importen más funciones de las que se necesitan y se consuman más recursos.

```
1 from FuncionesBasicas.sumar import *
2
3 a = sumar(3, 4)
4 print(a)
5
6 a = sumar_tres(3, 4, 5)
7 print(a)
```

```
7
12
Process finished with exit code 0
```

Python tiene sus propios módulos, los cuales forman parte de su librería de módulos estándar, que también pueden ser importados del mismo modo que los módulos definidos por el usuario.

1.3 Otras consideraciones

Aunque no es obligatorio, las normas de estilo de Python (PEP 8), nos recomiendan que a la hora de realizar las importaciones en nuestros programas, las importaciones de módulos se hagan al principio del documento, en orden alfabético de paquetes y módulos.

Además, nos recomiendan importar primero los módulos propios de Python, luego los módulos de terceros y por último los módulos definidos por el usuario, separando los bloques de importaciones con una línea en blanco.

1.4 Ejercicios propuestos

A continuación, se propone un ejercicio para aplicar los conceptos básicos de importación de funciones. La solución a dicho ejercicio, se encuentra al final del documento, pero se recomienda al alumno que trate de desarrollarlos.

1.4.1 Ejercicio 1

Basándote en el ejercicio 2 del tema 5, diseña la función en un archivo independiente del resto del programa. Además, dentro de ese fichero, diseña otra función que calcule las coordenadas del punto medio entre ambos puntos.

```
Introduce la coordenada x del primer punto: 7.1
Introduce la coordenada y del primer punto: 5
Introduce la coordenada x del segundo punto: 1.2
Introduce la coordenada y del segundo punto: 11
La distancia entre los dos puntos es 8.414867794564572
Las coordenadas del punto medio son: 4.1499999999999995, 8.0
```

2 Algunos módulos propios de Python

2.1 Módulo time

El módulo *time* está contenido dentro de la biblioteca estándar de Python nos va a proporcionar un conjunto de funciones para trabajar con fechas y/o horas, así como otras funciones en los sub-módulos *“datetime”* y *“calendar”*.

Las funciones que nos ofrece este módulo nos van a servir para obtener la fecha y/u hora de distintos tipos de relojes, información sobre nuestro huso horario, conversión de fechas y/o horas entre distintos formatos, validar y aplicar formatos o incluso para detener durante un tiempo la ejecución de un programa.

En general, el módulo *time* nos va a devolver el tiempo en segundos contados desde el 1 de enero de 1970 a las 0 horas, referido al tiempo civil del país en el que se ejecuta el programa (entendamos el que está configurado en el sistema operativo que ejecuta el programa). También podremos obtener el tiempo *“Tiempo Universal Coordinado”* (UTC) para todos los países en función del huso horario en el que se encuentre.

Para obtener el tiempo local en segundos utilizaremos la función *“time()”* que nos devuelve la cuenta de segundos desde la fecha indicada anteriormente.

```
1 import time
2
3 tiempo_segundos = time.time()
4 print(tiempo_segundos)
```

```
1524333022.496630212
```

```
Process finished with exit code 0
```

La función *“ctime(segundos)”* nos transforma este tiempo en segundos a una cadena estándar compuesta de *“día de la semana”, “mes”, “número de día de mes”, “hora”, “minuto”, “segundo”* y *“año”*.

```
1 import time
2
3 tiempo_segundos = time.time()
4 print(tiempo_segundos)
5
6 tiempo_cadena = time.ctime(tiempo_segundos)
7 print(tiempo_cadena)
```

```
1524333701.3843868
```

```
Sat Apr 21 20:01:41 2018
```

```
Process finished with exit code 0
```

La función *“gmtime()”* nos transforma el tiempo en segundos a un objeto de tipo *“struct_time”* que contiene el tiempo UTC. El resultado es una tupla con nueve valores enteros con los siguientes elementos: *“tm_year”, “tm_mon”, “tm_mday”, “tm_hour”, “tm_min”, “tm_sec”, “tm_wday”,*

“tm_yday” y “tm_isdst”, donde los valores corresponden al año, el mes, el día del mes, la hora, el minuto, el segundo, el número de día de la semana, el número de día del año y la vigencia o no del horario de verano (0 no vigente, 1 si vigente y -1 desconocido).

```
1 import time
2
3 tiempo_segundos = time.time()
4 print(tiempo_segundos)
5
6 tiempo_cadena = time.ctime(tiempo_segundos)
7 print(tiempo_cadena)
8
9 tiempo_estructurado = time.gmtime(tiempo_segundos)
10 print(tiempo_estructurado)
```

1524333942.1505687
Sat Apr 21 20:05:42 2018
time.struct_time(tm_year=2018, tm_mon=4, tm_mday=21, tm_hour=18, tm_min=5, tm_sec=42, tm_wday=5, tm_yday=111, tm_isdst=0)

Process finished with exit code 0

Podemos acceder a los valores individuales de la tupla que nos devuelve la función anterior, llamando al valor por su nombre, como si de una propiedad se tratase.

```
1 import time
2
3 tiempo_segundos = time.time()
4 tiempo_estructurado = time.gmtime(tiempo_segundos)
5 print(tiempo_estructurado)
6
7 print(tiempo_estructurado.tm_year)
8 print(tiempo_estructurado.tm_mon)
9 print(tiempo_estructurado.tm_day)
```

time.struct_time(tm_year=2018, tm_mon=4, tm_mday=21, tm_hour=18, tm_min=5, tm_sec=42, tm_wday=5, tm_yday=111, tm_isdst=0)
2018
4
21

Process finished with exit code 0

Obtendremos un objeto equivalente con el tiempo civil local, usando la función “localtime()”.

```
1 import time
2
3 tiempo_segundos = time.time()
4 tiempo_estructurado = time.gmtime(tiempo_segundos)
5 print(tiempo_estructurado)
6
7 tiempo_estructurado_local = time.localtime(tiempo_segundos)
```

```
8 print(tiempo_estructurado_local)

time.struct_time(tm_year=2018, tm_mon=4, tm_mday=21, tm_hour=18, tm_min=14,
tm_sec=50, tm_wday=5, tm_yday=111, tm_isdst=0)
time.struct_time(tm_year=2018, tm_mon=4, tm_mday=21, tm_hour=20, tm_min=14,
tm_sec=50, tm_wday=5, tm_yday=111, tm_isdst=1)

Process finished with exit code 0
```

También tenemos la función “*asctime()*” para convertir objetos de tipo “*struct_time*” a cadenas de texto y la función “*mktime()*” para pasarlo a segundos.

```
1 import time
2
3 tiempo_estructurado = time.localtime()
4 tiempo_cadena = time.asctime(tiempo_estructurado)
5 print(tiempo_cadena)
6
7 tiempo_segundos = time.mktime(tiempo_estructurado)
8 print(tiempo_segundos)

Sun Apr 22 13:51:47 2018
1524397907.0

Process finished with exit code 0
```

El módulo “*time*” también nos permite introducir fechas y validarlas a un formato concreto, es decir, si queremos que un usuario de nuestro programa pueda introducir una fecha, podemos especificar el formato a utilizar (por ejemplo “día/mes/año”) y que con la función “*strptime()*” se valide si se cumple el formato y si la fecha es válida. En caso de que no se cumpla el formato o la fecha no sea válida, nos devolverá un error.

Probamos con una fecha valida y todo funciona correctamente:

```
1 import time
2
3 tiempo_st = time.strptime("29 Feb 2016", "%d %b %Y")
4 print(tiempo_st)

time.struct_time(tm_year=2016, tm_mon=2, tm_mday=29, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=0, tm_yday=60, tm_isdst=-1)

Process finished with exit code 0
```

Probamos con una fecha no valida (2018 no fue año bisiesto) y nos devuelve un error.

```
5 tiempo_st = time.strptime("29 Feb 2018", "%d %b %Y")
6 print(tiempo_st)

Traceback (most recent call last):
  File
"C:/Users/fmari/OneDrive/Datos/CursoPython18/Ejemplos/FuncionesBasicas/Ejemplos_tim
e.py", line 5, in <module>
```

```
tiempo_st = time.strptime("29 Feb 2018", "%d %b %Y")
File "C:\Program Files\Python36\lib\strptime.py", line 559, in _strptime_time
    tt = _strptime(data_string, format)[0]
File "C:\Program Files\Python36\lib\strptime.py", line 528, in _strptime
    datetime_date(year, 1, 1).toordinal() + 1
ValueError: day is out of range for month

Process finished with exit code 1
```

Si utilizamos una abreviatura no reconocida como correcta, también obtendremos error:

```
7 tiempo_st = time.strptime("28 Febr 2018", "%d %b %Y")
8 print(tiempo_st)

Traceback (most recent call last):
  File
"C:/Users/fmari/OneDrive/Datos/CursoPython18/Ejemplos/FuncionesBasicas/Ejemplos_tim
e.py", line 7, in <module>
    tiempo_st = time.strptime("28 Febr 2018", "%d %b %Y")
  File "C:\Program Files\Python36\lib\strptime.py", line 559, in _strptime_time
    tt = _strptime(data_string, format)[0]
  File "C:\Program Files\Python36\lib\strptime.py", line 362, in _strptime
    (data_string, format))
ValueError: time data '28 Febr 2018' does not match format '%d %b %Y'

Process finished with exit code 0
```

Otros formatos válidos:

```
9 tiempo_st = time.strptime("22-04-2018", "%d-%m-%Y")
10 print(tiempo_st)

time.struct_time(tm_year=2018, tm_mon=2, tm_mday=28, tm_hour=0, tm_min=0,
tm_sec=0, tm_wday=2, tm_yday=59, tm_isdst=-1)

Process finished with exit code 0
```

Como vemos en los ejemplos anteriores, podemos definir varios formatos a utilizar utilizando abreviaturas precedidas del símbolo de tanto por ciento. El formato es un parámetro opcional, en caso de que no se especifique ninguno se utilizará el formato “%a %b %d %H:%M:%S %Y”, donde “%a” es el día de la semana, “%b” es el mes, “%d” es el día del mes, “%H” la hora, “%M” el minuto, “%S” el segundo y “%Y” el año. Las abreviaturas disponibles podemos encontrarlas en la página <http://pubs.opengroup.org/onlinepubs/009695399/functions/strptime.html>

La función “*strptime()*” nos permite obtener el tiempo con un formato de texto determinado o formatear un objeto de tipo “*struct_time*”.

```
1 import time
2
3 tiempo_cadena = time.strftime("%d-%m-%Y %H:%M", time.gmtime())
4 print(tiempo_cadena)
5
6 print(time.strftime("%d-%m-%Y %H:%M"))

22-04-2018 12:45
22-04-2018 14:45
```

```
Process finished with exit code 0
```

Este módulo también proporciona una serie de variables que pueden ser útiles, *“altzone”* nos proporciona la diferencia entre el huso local y el tiempo UTC en segundos en horario de verano, *“timezone”* nos da la misma diferencia en horario de invierno, *“tzname”* que nos devuelve una tupla con el nombre del huso horario local ordinario y el de verano si lo hay y *“daylight”* que nos devuelve un entero (1, 0, -1) que indica si el horario de verano está aplicado, no lo está o se desconoce.

```
1 import time
2
3 print(time.altzone)
4 print(time.timezone)
5 print(time.tzname)
6 print(time.daylight)
```

```
-7200
-3600
('Hora estándar romance', 'Hora de verano romance')
1
```

```
Process finished with exit code 0
```

Además de las horas oficiales, este módulo también nos permite interactuar con los relojes del sistema. La función *“monotonic()”* nos da el tiempo en segundos del reloj monotónico del sistema, *“perf_counter()”* nos devuelve el tiempo en segundos del contador de rendimiento del sistema y *“process_time()”* nos da el tiempo en segundos de la CPU para el proceso actual.

```
1 import time
2
3 print(time.monotonic())
4 print(time.perf_counter())
5 print(time.process_time())
```

```
497482.546
3.3107134355703715e-07
0.109375
```

```
Process finished with exit code 0
```

Por último, destacar la función *“sleep()”* que nos permite pausar la ejecución de un programa durante un número determinado de segundos. Esta función puede ser muy útil para hacer que nuestro programa haga pausas o esperas, por ejemplo, cuando hacemos descargas masivas de una web podemos poner pausas para que el servidor no nos bloquee.

2.2 Módulos de sistema

El módulo “os” de Python es el encargado de proporcionar funciones para interaccionar con el sistema operativo de nuestras máquinas. Nos va a proporcionar funciones para interactuar con el sistema de modo que una misma función con los mismos parámetros pueda ser usada sobre distintos sistemas operativos, sin tener que adaptar sus parámetros al entorno de ejecución. Algunas de las funciones más útiles son:

Método	Descripción
os.access(ruta, modo_acceso)	Nos dice si podemos acceder o no a un archivo o directorio.
os.getcwd()	Nos proporciona la ruta del directorio actual (en el que se encuentra el archivo que se está ejecutando).
os.chdir(ruta)	Permite cambiar el directorio de trabajo.
os.chroot()	Permite cambiar el directorio de trabajo al directorio raíz.
os.chmod(ruta, permisos)	Cambia los permisos de un archivo o directorio.
os.chown(ruta, permisos)	Cambia el propietario de un archivo o directorio.
os.mkdir(ruta, {permisos})	Permite crear un directorio.
os.makedirs(ruta, {permisos})	Permite crear directorios recursivamente, es decir, si introducimos la ruta de una subcarpeta, crea todas las carpetas necesarias para crearla.
os.remove(ruta)	Elimina un archivo.
os.rmdir(ruta)	Elimina un directorio.
os.removedirs(ruta)	Elimina recursivamente un directorio y todo su contenido.
os.rename(nombre, nuevo_nombre)	Permite renombrar un archivo.
os.system(comando)	Permite ejecutar un comando de sistema operativo, como si lo ejecutásemos desde símbolo de sistema.

Dentro de este módulo, también podremos acceder a las variables de entorno del sistema a través de la propiedad “*environ()*”, que nos devuelve un diccionario con todas las variables del sistema (en el ejemplo de código que podemos ver a continuación, se han eliminado los valores del diccionario y se han dejado sólo las claves para acortar la cadena de texto que devuelve. Además, no es recomendable publicar los datos que muestra por seguridad).

```

1  import os
2
3  print(os.environ)

environ({'ALLUSERSPROFILE', 'APPDATA', 'ASL.LOG', 'COMMONPROGRAMFILES',
'COMMONPROGRAMFILES(X86)', 'COMMONPROGRAMW6432', 'COMPUTERNAME', 'COMSPEC',
'FPS_BROWSER_APP_PROFILE_STRING', 'FPS_BROWSER_USER_PROFILE_STRING', 'GDAL_DATA',
'HOMEDRIVE', 'HOMEPATH', 'LOCALAPPDATA', 'LOGONSERVER', 'NUMBER_OF_PROCESSORS',
'ONEDRIVE', 'OS', 'PATH', 'PATHEXT', 'POSTGIS_ENABLE_OUTDB_RASTERS',
'POSTGIS_GDAL_ENABLED_DRIVERS', 'PROCESSOR_ARCHITECTURE', 'PROCESSOR_IDENTIFIER',
'PROCESSOR_LEVEL', 'PROCESSOR_REVISION', 'PROGRAMDATA', 'PROGRAMFILES',
'PROGRAMFILES(X86)', 'PROGRAMW6432', 'PSMODULEPATH', 'PUBLIC', 'PYCHARM_HOSTED',
'PYTHONIOENCODING', 'PYTHONPATH', 'PYTHONUNBUFFERED', 'SESSIONNAME', 'SYSTEMDRIVE',
'SYSTEMROOT', 'TEMP', 'TMP', 'USERDOMAIN', 'USERDOMAIN_ROAMINGPROFILE', 'USERNAME',
'USERPROFILE', 'WINDIR'})

```

Process finished with exit code 0

También dentro del módulo “os” tenemos el sub-módulo “path” que nos permite acceder a funciones relacionadas con los nombres de rutas y directorios. Entre ellos podemos destacar los siguientes:

Método	Descripción
os.path.abspath(ruta)	Devuelve la ruta absoluta del archivo.
os.path.basename(ruta)	Devuelve el directorio base de la ruta.
os.path.exists(ruta)	Nos dice si una ruta existe o no en disco.
os.path.getatime(ruta)	Devuelve el último acceso a un directorio.
os.path.getsize(ruta)	Devuelve el tamaño del archivo.
os.path.isabs(ruta)	Nos dice si una ruta es absoluta o no.
os.path.isfile(ruta)	Nos dice si una ruta es un archivo o no.
os.path.isdir(ruta)	Nos dice si una ruta es un directorio o no.

Otro módulo de sistema importante, es el módulo “sys” que nos proporciona funciones y variables directamente relacionadas con el intérprete del lenguaje. Entre las funciones de este módulo, podemos destacar:

Método	Descripción
sys.exit()	Fuerza la salida del interprete, es decir, cierra el programa que en ejecución.
sys.getdefaultencoding()	Devuelve el juego de codificación de caracteres por defecto.
sys.getfilesystemencoding()	Devuelve la codificación de caracteres que se usa para convertir los nombres de archivos Unicode en nombres de archivo del sistema.
sys.getsizeof(objeto, {valor_por_defecto})	Devuelve el tamaño de un determinado objeto. El segundo parámetro nos da la opción de establecer un valor de retorno por defecto e caso de que la función no devuelva nada.

Entre las variables que nos devuelve este módulo, podemos destacar:

Método	Descripción
sys.argv	Devuelve una lista con todos los argumentos que se han pasado por línea de comandos. Cuando ejecutamos un programa desde línea de comandos como por ejemplo “Python programa.py arg1 arg2”, se obtiene una lista con la forma [‘programa.py’, ‘arg1’, ‘arg2’]
sys.executable	Devuelve la ruta absoluta del ejecutable de Python que se está ejecutando.
sys.platform	Devuelve la plataforma sobre la cual se está ejecutando el intérprete.
sys.version	Devuelve la versión de Python con información adicional.

Por último, entre los módulos del sistema, destacamos el módulo “shutil” que nos permite copiar, mover o renombrar archivos y directorios. Destacamos las siguientes funciones:

Método	Descripción
shutil.copy(archivo_fuente, archivo_destino)	Permite copiar un archivo (El directorio debe existir).
shutil.copy2(archivo_fuente, archivo_destino)	Permite copiar un archivo manteniendo los metadatos del mismo (fecha de creación, modificación, etc.)
shutil.copymode(archivo_fuente, archivo_destino)	Copia los permisos de un archivo a otro archivo ya existente.

Método	Descripción
shutil.copystat(archivo_fuente, archivo_destino)	Copia los metadatos de un archivo a otro archivo ya existente.
shutil.move(archivo_fuente, archivo_destino)	Permite mover un archivo, de un directorio a otro.

2.3 Otros módulos

Como ya hemos dicho, Python nos proporciona gran cantidad de módulos integrados, los cuales están diseñados para resolver necesidades de programación. Por ejemplo, el módulo “*random*” nos permite obtener datos aleatorios, por ejemplo:

Método	Descripción
random.randint(a, b)	Devuelve un número entero aleatorio entre a y b.
random.choice(lista)	Devuelve un valor aleatorio de la lista que le hemos pasado.
random.shuffle(lista)	Devuelve la lista mezclada aleatoriamente.
random.sample(lista, n)	Devuelve una muestra de datos aleatoria de lista que le hemos pasado.

Ejemplos de código:

1	<code>import random</code>	
2		
3	<code># Generar una lista de 50 números aleatorios entre 0 y 1000</code>	
4	<code>lista = []</code>	
5		
6	<code>for n in range(0, 50):</code>	
7	<code> lista.append(random.randint(0, 1000))</code>	
8		
9	<code>print(lista)</code>	
		<code>[277, 449, 551, 882, 939, 81, 741, 739, 928, 958, 759, 791, 149, 547, 740, 992, 177, 260, 841, 809, 501, 704, 659, 19, 29, 915, 102, 919, 9, 982, 399, 861, 424, 409, 826, 501, 826, 893, 877, 328, 295, 1000, 427, 866, 395, 719, 270, 788, 371, 896]</code>
10	<code># Un valor aleatorio de la lista</code>	
11	<code>print(random.choice(lista))</code>	
		<code>919</code>
12	<code># Mezcla aleatoriamente la lista</code>	
13	<code>random.shuffle(lista)</code>	
14	<code>print(lista)</code>	
		<code>[741, 809, 739, 9, 395, 919, 740, 759, 81, 19, 882, 826, 788, 149, 547, 719, 371, 939, 551, 841, 896, 893, 260, 861, 409, 270, 704, 501, 295, 1000, 424, 982, 501, 399, 877, 928, 277, 427, 958, 992, 915, 659, 791, 29, 328, 177, 102, 449, 866, 826]</code>
15	<code># Diez valores aleatorios de la lista</code>	
16	<code>print(random.sample(lista, 10))</code>	
		<code>[896, 704, 371, 659, 915, 399, 861, 982, 882, 424]</code>

El módulo “*textwrap*” nos ofrece funciones para trabajar con textos y darles formato según nuestras necesidades, por ejemplo:

Método	Descripción
textwrap.wrap(texto, longitud)	Devuelve una lista con porciones de texto de como máximo la longitud establecida, utilizando los espacios para la separación.
textwrap.fill(texto, longitud)	Parecido al anterior, pero en lugar de devolver una lista, nos devuelve el mismo texto con saltos de línea insertados.
textwrap.indent(texto)	Permite añadir tabulaciones o espacios al principio de cada línea de texto.
textwrap.dedent(texto)	Permite quitar tabulaciones o espacios al principio de cada línea de texto (Si hay varias longitudes de tabulación, solo quita la más corta).
textwrap.shorten(texto, longitud)	Permite acortar un texto a una determinada longitud. Sustituye el resto del texto por “[...]”.

Ejemplos de código:

1	<code>import textwrap</code>
2	
3	<code>texto = "Esto es un texto de ejemplo más largo de lo que necesitamos"</code>
4	
5	<code># Dividir un texto en porciones de 15 caracteres</code>
6	<code>print(textwrap.wrap(texto, 15))</code>
	<code>'Esto es un', 'texto de', 'ejemplo más', 'largo de lo que', 'necesitamos']</code>
7	<code># Insertar saltos de línea a 15 caracteres</code>
8	<code>print(textwrap.fill(texto, 15))</code>
	<code>Esto es un texto de ejemplo más largo de lo que necesitamos</code>
9	<code># Añadiendo tabulaciones</code>
10	<code>texto2 = """Esto es</code>
11	<code>un texto</code>
12	<code>escrito en</code>
13	<code>varias líneas"""</code>
14	<code>print(textwrap.indent(texto, " "))</code>
	<code>Esto es un texto escrito en varias líneas</code>
15	<code># Quitando tabulaciones</code>
16	<code>texto3 = """ Esto es</code>
17	<code>un texto</code>
18	<code>escrito en</code>
19	<code>varias líneas</code>
20	<code>y tabulado"""</code>
21	<code>print(textwrap.dedent(texto))</code>

```
Esto es  
un texto  
escrito en  
varias líneas  
y tabulado
```

```
20 # Acortando texto  
21 print(textwrap.shorten(texto, 30))
```

```
Esto es un texto de [...]
```

2.4 Ejercicios propuestos

A continuación, se propone un ejemplo de ejercicio para aplicar conceptos básicos de utilización de funciones propias del lenguaje Python. La solución a dicho ejercicio, se encuentra al final del documento, pero se recomienda al alumno que trate de desarrollarlos.

2.4.1 Ejercicio 2

Diseña un programa que haga lo siguiente:

- Mostrar un mensaje de inicio con la hora de ejecución.
- Mostrar un mensaje con la ubicación del archivo que se está ejecutando.
- Pedir al usuario un número de elementos para una lista (numérica).
- Pedir el valor mínimo de la lista.
- Pedir el valor máximo de la lista.
- Generar una lista con el número de elementos pedidos entre el mínimo y el máximo con valores aleatorios.
- Mostrar los valores por pantalla.
- Mostrar un mensaje de finalización con la hora de ejecución.

```
Iniciando a las 21:00:00
```

```
Ejecutando el script C:/Users/fmari/OneDrive/Datos/CursoPython18/Ejemplos/Ejercicios/Listas1.py
```

```
Introduce la longitud de la lista: 500
```

```
Introduce el valor mínimo de la lista: 0
```

```
Introduce el valor máximo de la lista: 500
```

```
[266, 360, 388, 90, 46, 124, 288, 206, 203, 280, 453, 373, 314, 211, 63, 199, 312, 31, 291, 121,  
140, 289, 403, 140, 253, 469, 335, 63, 38, (...), 305, 421, 420, 182, 341, 309, 406, 27, 332,  
458, 429, 93, 137, 361, 84, 410, 80, 320, 261, 103, 371, 132, 106, 374]
```

3 Repositorios

Además de los módulos integrados que nos proporciona el lenguaje Python y de los módulos que nosotros podamos crear, también existe la posibilidad de utilizar módulos y paquetes de terceros. Para ello, acudiremos a repositorios de módulos, donde otros desarrolladores cuelgan sus paquetes para hacerlos accesibles a todo el mundo.

El repositorio más conocido es <https://pypi.org/> donde es muy sencillo encontrar miles de paquetes con funciones que nos pueden ayudar en el desarrollo de nuestros proyectos. Desde esta página web podemos buscar paquetes. Por ejemplo, vamos a buscar “*numpy*” que es un módulo muy interesante para realizar operaciones matemáticas.

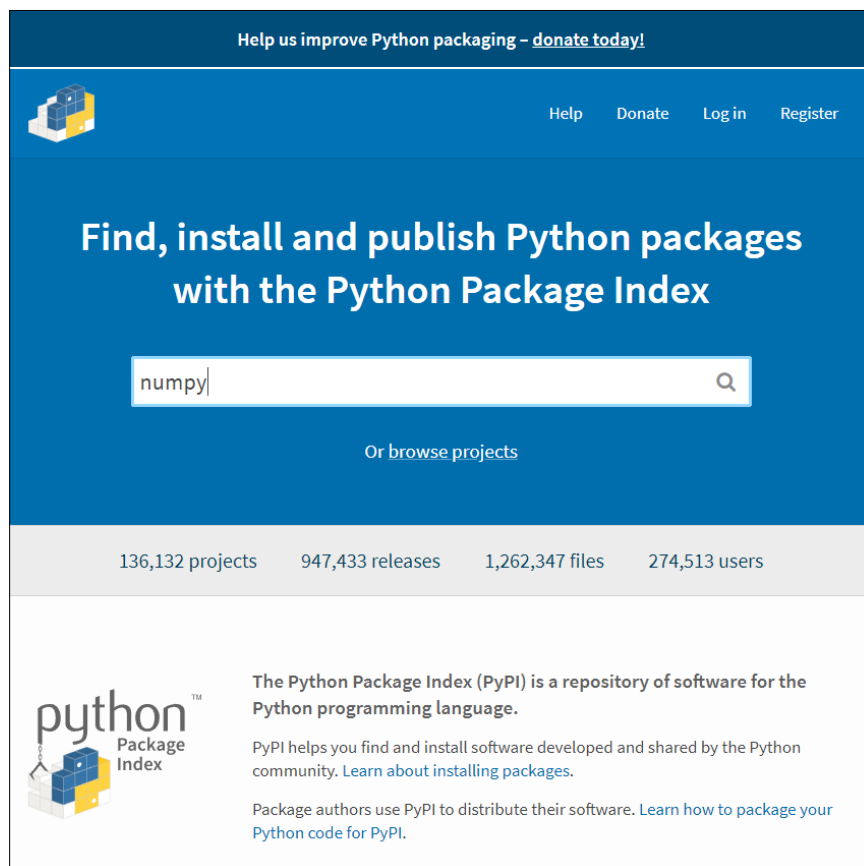


Imagen 5: Repositorio pypi.org.

Obtendremos una lista de resultados con todos los paquetes que lleven la palabra *numpy* en su nombre, en nuestro caso nos interesará el primer resultado.

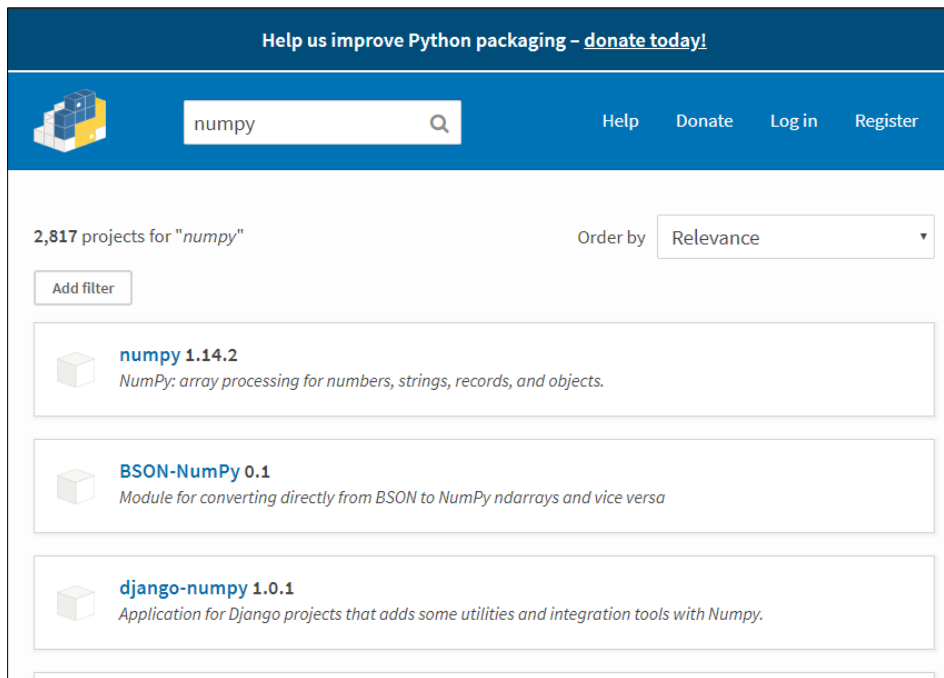


Imagen 6: Resultados de la búsqueda de numpy.

De este modo, accederemos a la página del paquete donde tendremos una completa descripción del mismo.

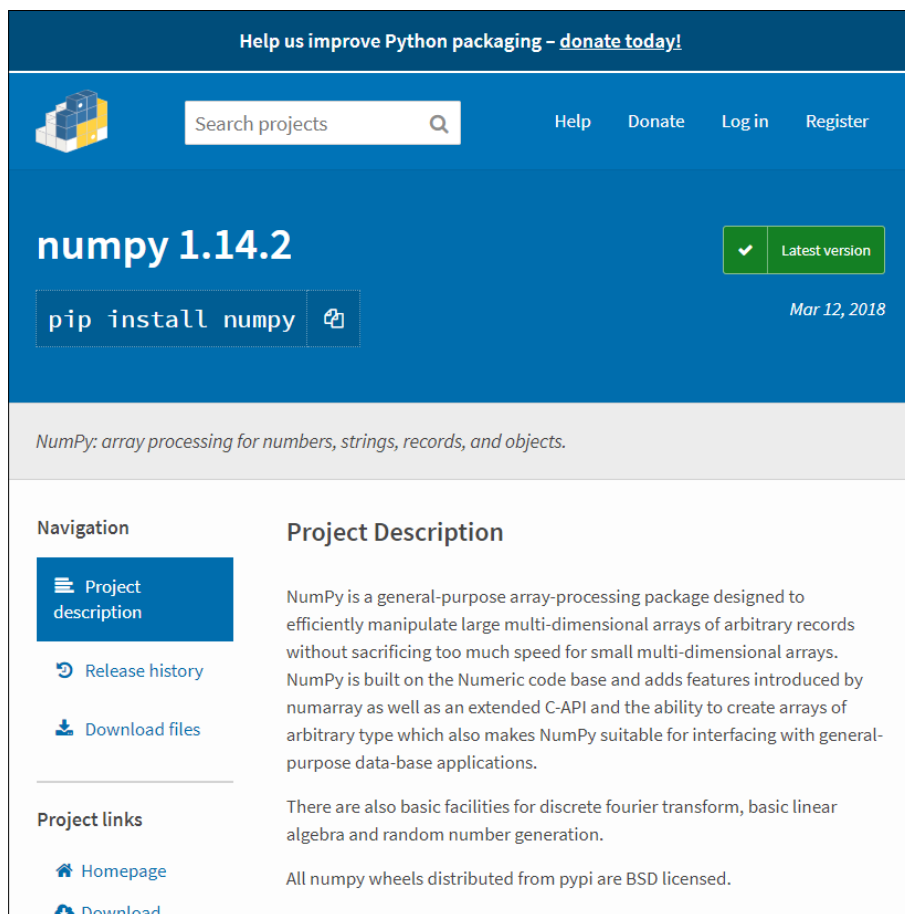


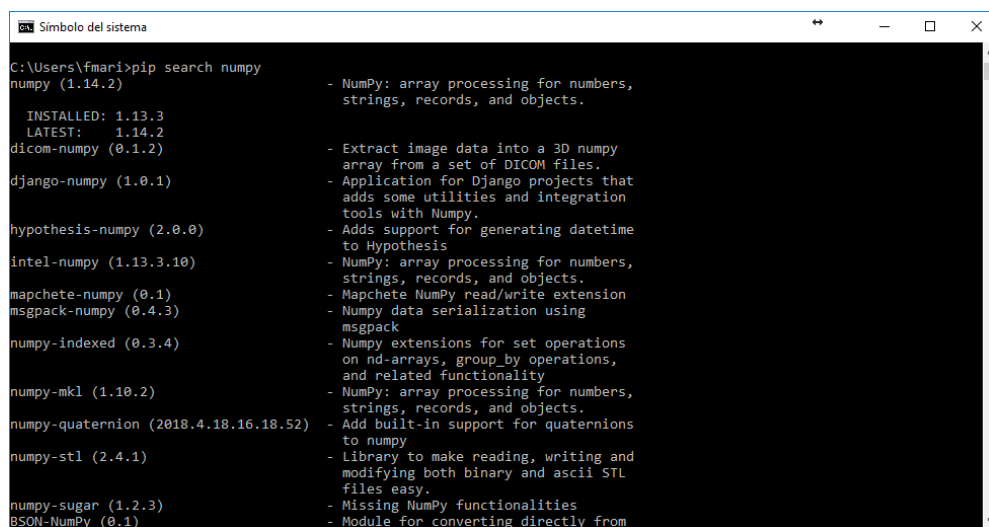
Imagen 7: Descripción de numpy.

En esta página, además de la descripción del paquete, también tenemos la opción de descargarlo, que nos llevará a una página donde podremos elegir el archivo a descargar de entre múltiples versiones disponibles en función de nuestro sistema operativo y nuestra versión de Python.

Sin embargo, hay otro camino para instalar los paquetes de este repositorio, que nos facilitarán el proceso, ya que Python trae instalado un gestor de paquetes que se conecta este repositorio y nos permite gestionar la instalación y mantenimiento de nuestros paquetes sin necesidad de preocuparnos por versiones. Además, en caso de que un determinado paquete necesite algún otro paquete para funcionar, también se encargará de instalarlo.

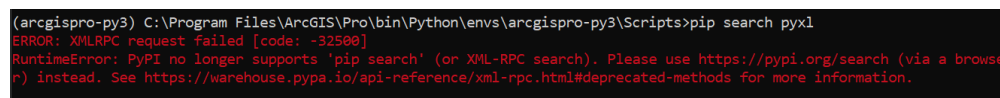
Se trata del gestor “*pip.exe*” que se encuentra en la carpeta “*Scripts*” de la ruta de instalación de nuestra versión de Python. Para poder utilizarlo vamos a abrir una ventana de símbolo de sistema (con privilegios de administrador) y llamaremos al comando “*pip*” más la instrucción que queramos ejecutar (en caso de que nos de error, deberemos o bien añadir la ruta de la carpeta “*Scripts*” al path del sistema o bien poner la ruta completa de “*pip*”).

En versiones previas se permitía buscar paquetes directamente, pero se cesó el soporte de esta opción por el excesivo tráfico que esto generaba. La instrucción era sencilla: “*pip search texto_a_buscar*”.



```
Símbolo del sistema
C:\Users\fmari>pip search numpy
numpy (1.14.2)
  INSTALLED: 1.13.3
  LATEST: 1.14.2
dicom-numpy (0.1.2)
  - Extract image data into a 3D numpy array from a set of DICOM files.
  - Application for Django projects that adds some utilities and integration tools with Numpy.
django-numpy (1.0.1)
  - Adds support for generating datetime to Hypothesis
hypothesis-numpy (2.0.0)
  - NumPy: array processing for numbers, strings, records, and objects.
intel-numpy (1.13.3.10)
  - NumPy: array processing for numbers, strings, records, and objects.
mapchete-numpy (0.1)
  - Mapchete NumPy read/write extension
msgpack-numpy (0.4.3)
  - Numpy data serialization using msgpack
numpy-indexed (0.3.4)
  - Numpy extensions for set operations on nd-arrays, group_by operations, and related functionality
numpy-mkl (1.10.2)
  - NumPy: array processing for numbers, strings, records, and objects.
numpy-quaternion (2018.4.18.16.18.52)
  - Add built-in support for quaternions to numpy
numpy-stl (2.4.1)
  - Library to make reading, writing and modifying both binary and ascii STL files easy.
numpy-sugar (1.2.3)
  - Missing NumPy functionalities
BSON-NumPy (0.1)
  - Module for converting directly from
```

Imagen 8: Búsqueda de numpy desde pip antes de su cese.

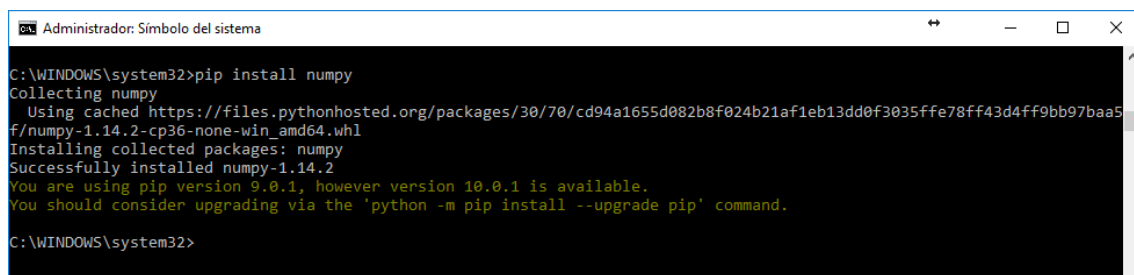


```
(arcgispro-py3) C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\Scripts>pip search pyx1
ERROR: XMLRPC request failed [code: -32500]
RuntimeError: PyPI no longer supports 'pip search' (or XML-RPC search). Please use https://pypi.org/search (via a browser) instead. See https://warehouse.pypa.io/api-reference/xml-rpc.html#deprecated-methods for more information.
```

Imagen 9: Resultado actual de pip search

Actualmente, aunque existen herramientas de terceros para las labores que hacía el comando search, se recomienda buscar en la web del proyecto Pypi.

Para instalar un paquete utilizaremos el comando “*pip install nombre_del_paquete*”. Se encargará de descargarlo e instalarlo.

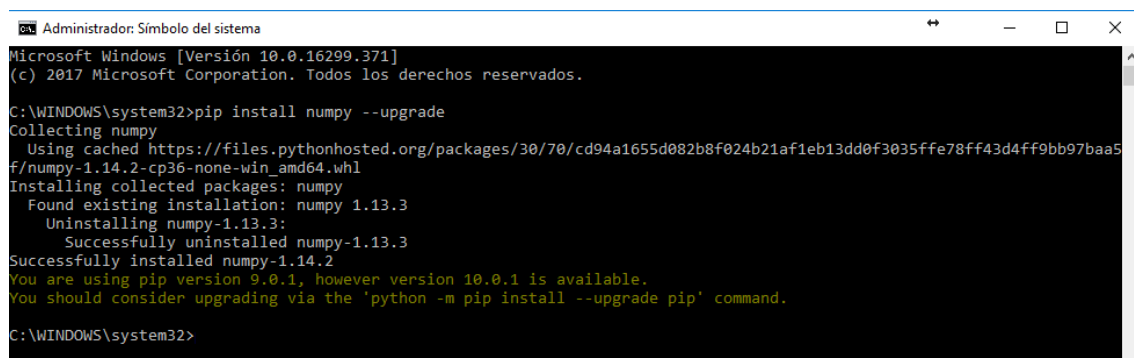


```

C:\WINDOWS\system32>pip install numpy
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/30/70/cd94a1655d082b8f024b21af1eb13dd0f3035ffe78ff43d4ff9bb97baa5f/numpy-1.14.2-cp36-none-win_amd64.whl
Installing collected packages: numpy
Successfully installed numpy-1.14.2
You are using pip version 9.0.1, however version 10.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
C:\WINDOWS\system32>
  
```

Imagen 10: Instalando numpy desde pip.

Para actualizar un paquete usaremos el comando “*pip install nombre_del_paquete --upgrade*”.



```

Microsoft Windows [Versión 10.0.16299.371]
(c) 2017 Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>pip install numpy --upgrade
Collecting numpy
  Using cached https://files.pythonhosted.org/packages/30/70/cd94a1655d082b8f024b21af1eb13dd0f3035ffe78ff43d4ff9bb97baa5f/numpy-1.14.2-cp36-none-win_amd64.whl
Installing collected packages: numpy
  Found existing installation: numpy 1.13.3
  Uninstalling numpy-1.13.3:
    Successfully uninstalled numpy-1.13.3
Successfully installed numpy-1.14.2
You are using pip version 9.0.1, however version 10.0.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
C:\WINDOWS\system32>
  
```

Imagen 11: Actualizando numpy desde pip.

Otros comandos que nos proporciona este gestor son:

Comando	Descripción
pip freeze	Devuelve una lista de todos los paquetes instalados que cumplen los requerimientos de pip.
pip uninstall nombre_paquete	Desinstala un paquete determinado
pip download	Descarga un paquete pero no lo instala.
pip list	Devuelve una lista con todos los paquetes instalados.
pip show nombre_paquete	Devuelve la información sobre un determinado paquete.
pip check nombre_paquete	Comprueba la existencia de paquetes dependientes.

4 Soluciones a los ejercicios

Nota: Cómo en todos los problemas de programación, las soluciones no son únicas. En este apartado se plantean posibles soluciones. Al tratarse de ejemplos de carácter didáctico no siempre serán las soluciones óptimas.

4.1 Ejercicio 1

Basándote en el ejercicio 2, diseña la función en un archivo independiente del resto del programa. Además, dentro de ese fichero, diseña otra función que calcule las coordenadas del punto medio entre ambos puntos.

“Funciones.py”

```
1 def distancia(x1, y1, x2, y2):
2     dx = x2 - x1
3     dy = y2 - y1
4     dist = (dx**2 + dy**2)**0.5
5     return dist
6
7 def puntoMedio(x1, y1, x2, y2):
8     x_media = (x1 + x2) / 2
9     y_media = (y1 + y2) / 2
10    return (x_media, y_media)
```

En primer lugar, creamos un archivo independiente al que hemos llamado “Funciones.py” y en el que escribiremos las funciones de distancia y de punto medio.

```
1 import Funciones
2
3 x1 = float(input('Introduce la coordenada x del primer punto: '))
4 y1 = float(input('Introduce la coordenada y del primer punto: '))
5 x2 = float(input('Introduce la coordenada x del segundo punto: '))
6 y2 = float(input('Introduce la coordenada y del segundo punto: '))
7
8 dist = Funciones.distancia(x1, y1, x2, y2)
9
10 print('La distancia entre los dos puntos es', dist)
11
12 xm, ym = Funciones.puntoMedio(x1, y1, x2, y2)
13
14 print('Las coordenadas del punto medio son: ' + str(xm) + ', ' + str(ym))
```

```
Introduce la coordenada x del primer punto: 7.1
Introduce la coordenada y del primer punto: 5
Introduce la coordenada x del segundo punto: 1.2
Introduce la coordenada y del segundo punto: 11
La distancia entre los dos puntos es 8.414867794564572
Las coordenadas del punto medio son: 4.1499999999999995, 8.0
```

```
Process finished with exit code 0
```

A continuación, escribimos nuestro programa principal. En la línea 1 importamos el módulo de funciones, entre las líneas 3 y 6 pedimos el valor de las coordenadas, en la línea 8 llamamos a la función de distancia para imprimir su resultado en la línea 10 y en la línea 12 llamamos a la función que calcula el punto medio para imprimir su resultado en la línea 14.

4.2 Ejercicio 2

Diseña un programa que haga lo siguiente:

- Mostrar un mensaje de inicio con la hora de ejecución.
- Mostrar un mensaje con la ubicación del archivo que se está ejecutando.
- Pedir al usuario un número de elementos para una lista (numérica).
- Pedir el valor mínimo de la lista.
- Pedir el valor máximo de la lista.
- Generar una lista con el número de elementos pedidos entre el mínimo y el máximo con valores aleatorios.
- Mostrar los valores por pantalla.
- Mostrar un mensaje de finalización con la hora de ejecución.

```
1 import random
2 import sys
3 import time
4
5 print('Iniciando a las', time.strftime("%H:%M:%S"))
6 print('Ejecutando el script', sys.argv[0])
7
8 longitud = int(input('Introduce la longitud de la lista: '))
9 min = float(input('Introduce el valor mínimo de la lista: '))
10 max = float(input('Introduce el valor máximo de la lista: '))
11
12 lista = []
13 for n in range(0, longitud):
14     lista.append(random.randint(min, max))
15
16 print(lista)
17 print('Finalizado a las', time.strftime("%H:%M:%S"))
```

```
Iniciando a las 21:00:00
Ejecutando el script
C:/Users/fmari/OneDrive/Datos/CursoPython18/Ejemplos/Ejercicios/Listas1.py
Introduce la longitud de la lista: 500
Introduce el valor mínimo de la lista: 0
Introduce el valor máximo de la lista: 500
[266, 360, 388, 90, 46, 124, 288, 206, 203, 280, 453, 373, 314, 211, 63, 199, 312, 31,
291, 121, 140, 289, 403, 140, 253, 469, 335, 63, 38, (...), 305, 421, 420, 182, 341,
309, 406, 27, 332, 458, 429, 93, 137, 361, 84, 410, 80, 320, 261, 103, 371, 132, 106,
374]
Finalizado a las 21:00:09

Process finished with exit code 0
```

Importamos los módulos necesarios para desarrollar lo que se pide en el ejercicio y vamos utilizándolos. Este ejemplo nos sirve para ver la sencillez de uso de las librerías de Python.

5 Bibliografía

Bahit, E. (s.f.). *Python para principiantes*. Obtenido de <https://librosweb.es/libro/python/>

Foundation, P. S. (2013). *Python.org*. Obtenido de <https://www.python.org/>

Suárez Lamadrid, A., & Suárez Jiménez, A. (Marzo de 2017). *Python para impacientes*. Obtenido de <http://python-para-impacientes.blogspot.com.es/2017/03/el-modulo-time.html>