Capítulo 5

Módulos

math

random

time

datetime

...

Hasta este momento hemos estado usando las palabras claves que forman el esqueleto básico de Python, así como algunas otras funciones específicas que hacen algunas tareas también de carácter general. Sin embargo, Python es un lenguaje de programación tremendamente rico en funcionalidades disponibles a través de módulos que tenemos a nuestra disposición para usar en cualquier momento. Si se quiere obtener un listado exhaustivo de los módulos disponibles, lo más adecuado es consultar la documentación en la web oficial de Python (www.python.org).

Por defecto, cualquier instalación de Python ya trae un elevado número de módulos que nos permitirán realizar todo tipo de tareas, pero además podemos instalar módulos desarrollados por terceros para añadir aún más funcionalidad a nuestra instalación.

Para comenzar y familiarizarnos con el uso de los módulos vamos a ver qué son, como se usan y cuales son los más habituales cuando uno se está iniciando con Python.

¿Qué son los módulos?

Módulos, librerías, bibliotecas, …, estos términos suelen ser muy parecidos y su objetivo es poner a disposición del programador una serie de herramientas que le faciliten la realización de algunas tareas específicas.

Los módulos suelen ser colecciones de funciones, constantes, clases, objetos, ... que tenemos agrupados bajo un mismo nombre.

Ejemplo:

En el módulo math, hay disponible funciones para la realización de cálculos matemáticos como logaritmos, exponenciales, raíces, redondeos, trigonometría, m.c.d., etc.

¿Cómo se usan los módulos?

Tenemos 2 formas de usar los módulos, dependiendo del número de funciones que vayamos a usar y dependiendo de que no existan interferencias entre diferentes módulos que puedan coincidir los nombres de algunas de las funciones que contengan.

Forma 1: import math

Al usar el módulo math de esta manera estamos importando **todas las funciones** que contiene el módulo y por tanto a partir de este momentos todas están disponibles **anteponiéndoles el nombre del módulo**.

Ejemplo:

math.sqrt(9)	De esta forma podríamos calcular la raíz cuadrada de 9.
math.gcd(60,48)	Calculamos el M.C.D. de 60 y 48.

Forma 2: from math import sqrt, gcd

De esta segunda forma, sólo importamos las funciones sqrt y gcd, es decir, sólo importamos las funciones enumeradas en la importación. Además, en este caso no es necesario anteponer el nombre del módulo para poder hacer uso de la función.

Ejemplo:

sqrt(9)	De esta forma podríamos calcular la raíz cuadrada de 9.
math.gcd(60,48)	Calculamos el M.C.D. de 60 y 48.

Una variante de esta segunda forma es: **from math import *** así estamos importando todas las funciones y podemos usarlas directamente sin necesidad de anteponer el nombre del módulo.

Módulos comunes y de uso frecuente

Hablar de módulos frecuentes no tiene mucho sentido puesto que se usan los módulos en función de las tareas que se quieren hacer, por lo tanto, la expresión módulos comunes sólo va a tener sentido en el ámbito de la aplicación que le daremos, que en nuestro caso es un uso académico en los niveles que trata este libro.

Como ya se adelanta en el nombre del capítulo, trabajaremos con los módulos: math, random, time y datetime. Más adelante veremos otros módulos en función de las necesidades que vayan surgiendo.

El módulo math

```
import math
math.
```

```
acos
acosh
asin
asinh
atan
atan2
atanh
```

Como su propio nombre indica, incluye funciones para realizar operaciones matemáticas.

Podemos encontrar una amplia descripción de cada una de las funciones y constantes que incluye este módulo en la documentación de Python, nosotros vamos a usar algunas de ellas que son bastante habituales.

Programa: operaciones-varias1.py

Vamos a hacer un programa que nos genere una tabla con valores y su raíz cuadrada (sqrt). Calcularemos esta operación para los número del 1 al 10.

1º Realizaremos la importación del módulo

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Generaremos un bucle para construir la tabla

4º Realizaremos los cálculos necesarios

```
1     import math
2     print('Este programa calculará las raíz cuadrada,')
3     print('de los números: 1 al 10')
4     for x in range(1,11):
5         print(x,math.sqrt(x))
```

La salida del programa debe ser algo similar a la siguiente imagen:

```
>>> %Run operaciones-varias1.py

  Este programa calculará las raíz cuadrada,
  de los números: 1 al 10
  1 1.0
  2 1.4142135623730951
  3 1.7320508075688772
  4 2.0
  5 2.23606797749979
  6 2.449489742783178
  7 2.6457513110645907
  8 2.8284271247461903
  9 3.0
  10 3.1622776601683795

>>>
```

Programa: mcd-mcm.py

Vamos a hacer un programa que nos calcule el máximo común divisor y el mínimo común múltiplo de 2 números que le introduzcamos al programa.

Nota: Vamos a recordar una propiedad sobre mcm y mcd, que es la siguiente:

mcm(x,y)*mcd(x,y)=x*y

1º Realizaremos la importación del módulo

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Solicitará los números

4º Realizaremos los cálculos necesarios

5º Mostraremos los resultados

```
1     import math
2     print('Este programa calculará el mcd y mcm,')
3     print('de los 2 números que introduzcamos')
4     n1=input('Escribe el número 1: ')
5     n1=eval(n1)
6     n2=input('Escribe el número 2: ')
7     n2=eval(n2)
8     mcd=math.gcd(n1,n2)
9     mcm=(n1*n2)/mcd
10    print('El Máximo Común Divisor es',mcd)
11    print('El Mínimo Común Múltiplo es',mcm)
```

Este programa generará la siguiente salida:

```
>>> %Run mcd-mcm.py

  Este programa calculará el mcd y mcm,
  de los 2 números que introduzcamos
  Escribe el número 1: 60
  Escribe el número 2: 48
  El Máximo Común Divisor es 12
  El Mínimo Común Múltiplo es 240.0

>>>
```

La librería math pone a disposición algunas constantes como son los números pi y e.

La forma de acceder a ellas es anteponiendo el nombre del módulo, es decir, math.pi y math.e nos dará los valores que observamos en la imagen:

```
>>> math.pi
3.141592653589793
>>> math.e
2.718281828459045
>>>
```

Programa: suma-iterable.py

Vamos a hacer un programa que nos calcule la suma de un conjunto de valores. Para simplificar el programa vamos a suponer que los valores ya los tenemos datos previamente.

Este programa se puede hacer con un bucle y un acumulador: suma = suma + valor, pero vamos a usar la función fsum del módulo math que facilita esta operación.

1º Realizaremos la importación de la función fsum del módulo math

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Realizaremos los cálculos necesarios

4º Mostraremos los resultados

```
1    from math import fsum
2    print('Este programa realiza la suma de un conjunto de valores')
3    valores=(2,4,12,5,31,45,32,13,15)
4    print(valores)
5    suma=fsum(valores)
6    print('La suma de los números es',suma)
```

Observa que en esta ocasión sólo hemos importado la función fsum del módulo math. Al hacer la importación de este modo, podemos usar directamente la función sin necesidad de anteponer el nombre del módulo.

La salida de nuestro programa debería ser algo similar a:

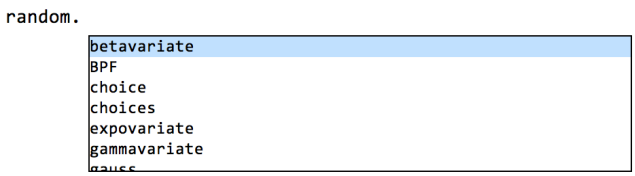
```
>>> %Run suma-iterable.py

  Este programa realiza la suma de un conjunto de valores
  (2, 4, 12, 5, 31, 45, 32, 13, 15)
  La suma de los números es 159.0

>>>
```

El módulo random

El módulo random nos permite disponer de herramientas para la generación de resultados aleatorios dentro de nuestros programas.

El uso de valores aleatorios es algo muy frecuente en la programación puesto que permite que el programa reaccione de diferentes formas ante una misma situación. Estas herramientas son fundamentales en la elaboración de juegos, simulación de escenarios, pruebas, generación de datos, etc.

Como ya dijimos con el módulo math, para conocer todas las funciones disponibles en el módulo random, lo mejor es consultar la documentación en la web www.python.org.Nosotros veremos unos ejemplos con algunas de las funciones.



Veamos con un ejemplo algunas de las funciones que resultan útiles dentro del módulo random.

Random.random()	Genera un número decimal aleatorio entre 0 y 1.
random.randint(a,b)	Genera un número entero aleatorio entre a y b.
random.uniform(a,b)	Genera un número decimal aleatorio entre a y b.
random.choice(conjunto)	Elige un elemento del conjunto pasado como argumento.

Programa: aleatorio1.py

Vamos a hacer un programa que genere un número decimal entre 0 y 1; un número entero entre 10 y 100; un número decimal entre 10 y 20; y por último elija un nombre de una lista de nombres que le pasaremos como parámetros.

1º Realizaremos la importación del módulo random

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Realizaremos realizaremos la generación que hemos indicado en el párrafo anterior

4º La lista es nombres=['Arturo','Julio','Dani','Aurora','Roberto','Martina','Hugo']

5º Mostraremos los resultados

```
1     import random
2     print('Programa para generar algunos datos aleatorios')
3     v1=random.random()
4     print('Valor aleatorio entre 0 y 1:',v1)
5     v2=random.randint(10,100)
6     print('Valor entero aleatorio entre 10 y 100:',v2)
7     v3=random.uniform(10,20)
8     print('Valor decimal entre 10 y 20:',v3)
9     nombres=['Arturo','Julio','Dani','Aurora','Roberto','Martina','Hugo']
10    nom=random.choice(nombres)
11    print('El nombre elegido es:',nom)
```

El programa debe realizar una salida similar a esta:

```
>>> %Run aleatorio1.py

   Programa para generar algunos datos aleatorios
   Valor aleatorio entre 0 y 1: 0.6803400712801796
   Valor entero aleatorio entre 10 y 100: 21
   Valor decimal entre 10 y 20: 12.346800023462455
   El nombre seleccionado es: Julio

>>>
```

Programa: aleatorio2.py

Vamos a hacer un programa que genere números aleatorios enteros entre 1 y 10, los vaya imprimiendo y finalice cuando salga el número 10.

1º Realizaremos la importación de la función randint del módulo random

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Realizaremos un bucle que se ejecutará mientras la generación sea distinta de 10

4º Dentro del mismo bucle iremos mostrando los valores generados

```
1     from random import randint
2     print('Programa que generará número aleatorios')
3     print('entre 0 y 10 hasta que salga el número 10')
```

```
4       n=0
5       while n!=10:
6           n=randint(1,10)
7           print('Ha salido el número:',n)
```

```
>>> %Run aleatorio2.py

  Programa que generará número aleatorios
  entre 0 y 10 hasta que salga el número 10
  Ha salido el número: 5
  Ha salido el número: 3
  Ha salido el número: 9
  Ha salido el número: 4
  Ha salido el número: 5
  Ha salido el número: 3
  Ha salido el número: 5
  Ha salido el número: 7
  Ha salido el número: 5
  Ha salido el número: 3
  Ha salido el número: 10

>>>
```

El programa debería mostrar una salida similar a esta:

El módulo time

El módulo time nos va a servir para manejar fechas, horas, medir tiempos, etc.

Generalmente en la mayoría de los programas es necesario introducir datos relacionados con el momento, la fecha o la hora en el que se producen determinados actos. Para estas ocasiones podemos usar este módulo que aunque tiene muchas opciones, las más comunes las vamos a utilizar en los siguientes ejemplos.

Antes de comenzar vamos a aclarar una cuestión. Python al igual que otros muchos lenguajes de programación utiliza el concepto de marca de tiempo (timestamp), con na precisión de una millonésima de segundo (no está nada mal), para esto se basa en el reloj interno que tienen nuestros equipos. Esta instante (timestamp), Python lo gestiona internamente como un valor numérico.

```
>>> import time
>>> time.time()
1516882907.320448
```

En el momento del diseño de Python, se acordó gestionar establecer el momento 0 en el **1 de enero de 1970 as las 00:00:00**. Por lo tanto, el número que obtenemos al ejecutar la función time() del módulo time, representa la cantidad de segundos transcurridos desde el momento 0. La precisión llega hasta la millonésima parte de un segundo.

También podemos obtener el momento actual con una cadena de texto y como una estructura de tiempo.

Veamos con un ejemplo como obtener la fecha y momento actual de cada una de estas formas:

Programa: tiempos1.py

Vamos a hacer un programa que mostrará la fecha y hora en diferentes formatos.

1º Realizaremos la importación del módulo time

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Vamos a utilizar las funciones: time, ctime, gmtime y localtime.

4º Dentro del mismo bucle iremos mostrando los valores generados

```
1       import time
2       print('Obtener la fecha y hora actual en diferentes formatos')
3       print('Tiempo en segundos:',time.time())
4       print('Fecha y hora como un texto:',time.ctime())
5       print('Estructura tiempo (UTC):',time.gmtime())
6       print('Estructura tiempo (local):',time.localtime())
```

Obtenemos el siguiente resultado:

```
>>> %Run tiempos1.py

  Obtener la fecha y hora actual en diferentes formatos
  Tiempo en segundos: 1516896310.2412071
  Fecha y hora como un texto: Thu Jan 25 17:05:10 2018
  Estructura tiempo (UTC): time.struct_time(tm_year=2018, tm_mon=1, tm_mday=25, tm_hour=16, tm_min=5,
  tm_sec=10, tm_wday=3, tm_yday=25, tm_isdst=0)
  Estructura tiempo (local): time.struct_time(tm_year=2018, tm_mon=1, tm_mday=25, tm_hour=17, tm_min=5
  , tm_sec=10, tm_wday=3, tm_yday=25, tm_isdst=0)

>>>
```

Como puedes observar en el ejemplo, las funciones gmtime() y localtime() sólo diferencian la hora dependiendo del huso horario en el que nos encontremos. UTC es el huso horario en el meridiano de Greenwich o en el meridiano cero.

Generalmente vamos a preferir tener la fecha y hora en un formato diferente, para ello el módulo time de Python facilita una función strftime(), que permite varios parámetros para construir cualquier cadena de texto representando la fecha. Por ejemplo:

strftime('%d/%m/%y',localtime()) nos serviría para obtener la fecha en el formato habitual 25/01/18

%d	Para mostrar los días
%m	Para mostrar el mes
%y	Para mostrar el año (2 dígitos)
%Y	Para mostrar el año (4 dígitos)
%H	Para mostrar las horas
%M	Para mostrar los minutos
%S	Para mostrar los segundos

Existen otras muchas opciones que nos permiten construir formatos personalizados de fechas, horas, fecha y hora.

De todas formas, con estos valores tendremos cubiertas la mayoría de los casos que se suelen necesitar.

Programa: tiempos2.py

Vamos a hacer un programa que mostrará la fecha y hora en varios formatos habituales.

1º Realizaremos la importación de las funciones strftime, gmtime, localtime del módulo time

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Construiremos los diferentes formatos

4º Mostraremos los formatos construidos

```
1   from time import gmtime,localtime,strftime
2   print('Fecha y hora en formatos habituales')
3   f_corto=strftime('%d/%m/%y',localtime())
4   print('Fecha en formato corto:',f_corto)
5   fh_corto=strftime('%d/%m/%y %H:%M:%S',localtime())
6   print('Fecha y Hora:',fh_corto)
7   f_sql=strftime('%Y-%m-%d',localtime())
8   print('Fecha SQL:',f_sql)
9   fh_sql=strftime('%Y-%m-%d %H:%M:%S',localtime())
10  print('Fecha y Hora SQL:',fh_sql)
```

```
>>> %Run tiempos2.py

   Fecha y hora en formatos habituales
   Fecha en formato corto: 25/01/18
   Fecha y Hora: 25/01/18 17:32:01
   Fecha SQL: 2018-01-25
   Fecha y Hora SQL: 2018-01-25 17:32:01

>>>
```

A continuación vamos a desarrollar un programa que nos pida introducir una palabra y nos indique el tiempo que hemos tardado en introducir la palabra.

Programa: tiempos3.py

1º Realizaremos la importación de la función time del módulo time

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Tomaremos la marca de tiempo antes de solicitar la entrada de texto

4º Solicitaremos la entrada de texto

5º Tomaremos luna nueva marca de tiempo después de la entrada de texto

4º Mostraremos el resultado

```
1    from time import time
2    print('Calcularé el tiempo que tardas en escribir una palabra')
3    t1=time()
4    palabra=input('Escribe una palabra y pulsa intro: ')
5    t2=time()
6    tiempo=int(t2-t1)
7    print('En escribir la palabra',palabra,'has tardado',tiempo,'segundos')
```

El resultado de este programa sería:

```
>>> %Run tiempos3.py

   Calcularé el tiempo que tardas en escribir una palabra
   Escribe una palabra y pulsa intro: Informática
   En escribir la palabra Informática has tardado 4 segundos

>>>
```

En algunas ocasiones queremos nuestro programa espere un determinado tiempo, por ejemplo cuando mostramos alguna infromación, es posible que sea necesario dar tiempo al usuario para leerla.

Dentro del módulo time, disponemos de una función muy útil para este tipo de tareas, se trata de la función **sleep()**. A esta función hay que indicarle mediante un parámetro, el número de segundos que deseamos que espere:

sleep(2), esperará 2 segundos para continuar con la ejecución del programa.

Vamos a hacer un programa que genere 10 números aleatorios entre 1 y 100. Esta tarea es inmediata, pero nuestro programa lo hará a lo largo de 10 segundos, generando un número cada segundo.

Programa: tiempos4.py

1º Realizaremos la importación de las funciones sleep y randint de los módulos time y random

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Crearemos un bucle for para las 10 repeticiones.

4º Generaremos 1 número y mostramos la información.

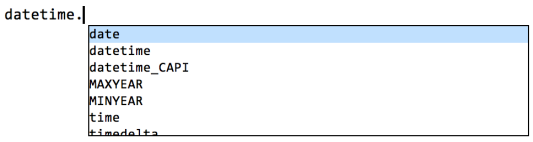
5º Esperamos 1 segundo antes de continuar con el bucle.

```
1    from random import randint
2    from time import sleep
3
4    print('Programa que genera 10 nùmeros aleatorios')
5    for n in range(10):
6        numero=randint(1,100)
7        print('Generación:',n+1,'Número generado:',numero)
8        sleep(1)
```

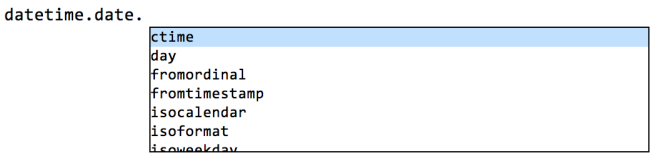
Este programa generará una salida similar a la siguiente imagen:

```
>>> %Run tiempos4.py

  Programa que genera 10 nùmeros aleatorios
  Generación: 1 Número generado: 79
  Generación: 2 Número generado: 66
  Generación: 3 Número generado: 55
  Generación: 4 Número generado: 44
  Generación: 5 Número generado: 1
  Generación: 6 Número generado: 87
  Generación: 7 Número generado: 38
  Generación: 8 Número generado: 54
  Generación: 9 Número generado: 33
  Generación: 10 Número generado: 38

>>>
```

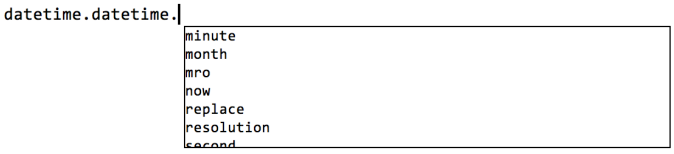
El módulo datetime



Dentro de las librerías estándar de Python también tenemos otro módulo que nos permite gestionar fechas de una forma un poco más fácil, este es el módulo datetime.



Este módulo pone a disposición varios objetos (clases) que disponen de muchas características, se trata de **datetime.date, datetime.time** y **datetime.datetime**:



Tanto el objeto date, como el datetime disponen del método today().

datetime.date.today() nos suministra la fecha actual.

datetime.datetime.today() nos suministra la fecha y la hora actual.

Veamos un ejemplo con estos objetos.

Programa: fechas1.py

1º Realizaremos la importación del módulo datetime.

2º El programa mostrará un mensaje indicando su funcionalidad.

3º Mostraremos la fecha actual y algunos datos más.

4º Mostraremos la fecha y hora actual y algunos datos más.

```
1     import datetime
2     print(40*'-')
3     print('Información con datetime.date')
4     hoy=datetime.date.today()
5     print(hoy)
6     print(hoy.weekday())
7     print(hoy.day)
8     print(40*'-')
9     print('Información con datetime.datetime')
10    ahora=datetime.datetime.today()
11    print(ahora)
12    print(ahora.hour)
13    print(ahora.minute)
```

El programa genera una salida similar a la imagen:

```
>>> %Run fechas1.py

  ----------------------------------------
  Información con datetime.date
  2018-01-28
  6
  28
  ----------------------------------------
  Información con datetime.datetime
  2018-01-28 11:07:30.880630
  11
  7

>>>
```

La función weekday() devuelve un valor numérico que indica el día de la semana teniendo en cuenta que 0=lunes, 1=martes, 2=miércoles, 3=jueves, 4=viernes, 5=sábado y 6=domingo.

Ejercicios:

Nota: Recuerda que para realizar los siguientes ejercicios debes importar los módulos que correspondan en cada caso.

Ejercicio 1:

Realizar un programa que obtenga la raíz cuadrada de todos los números impares entre 1 y 100.

Ejercicio 2:

Realizar un programa que obtenga la raíz cuadrada de todos los números pares entre 1 y 100.

Ejercicio 3:

Realizar un programa que obtenga el logaritmo en base 10 de todos los números entre 1 y 100. **Nota:** log10(x) sirve para calcular el logaritmo en base 10.

Ejercicio 4:

Realiza un programa que solicite una palabra y a continuación muestre un carácter aleatorio de los que contenga la palabra introducida.

Ejercicio 5:

Realiza un programa que tenga 10 nombres escritos en una lista, y que elija aleatoriamente uno de los nombres de la lista. **Nota:** usa el módulo random y la función **choice** de este módulo.

Ejercicio 6:

Realiza un programa que tenga 10 nombres escritos en una lista, y que elija aleatoriamente tres de los nombres de la lista. **Nota:** usa el módulo random y la función **sample** de este módulo.

Ejercicio 7:

Realiza un programa que tenga 10 nombres escritos en una lista, y que muestre los nombres desordenados, es decir, desordenarlos aleatoriamente. **Nota:** usa el módulo random y la función **suffle** de este módulo.

Ejercicio 8:

Realiza un programa que genere 6 números aleatorios entre 1 y 49. Como el juego de la primitiva.

Ejercicio 9:

Realiza un programa que genere 15 resultados 1, x o 2. Como una apuesta de la quiniela.

Ejercicio 10:

Realiza un programa que genere 2 número aleatorios entre 1 y 6 simulando el lanzamiento de 2 dados. Debe mostrar los 2 números obtenidos y la suma de dichos números.