LIBRERÍA ESTÁNDAR DE PYTHON

Actualmente Python incorpora una enorme librería por defecto, es la llamada Librería Estándar y podemos ver todos los detalles en su documentación

Ya conocemos algunos de estos módulos como *os*, *time* o *random*. Vamos a ver una descripción de los principales módulos:

- **os** todo lo relacionado con el sistema operativo (en general): ficheros, hilos, variables del entorno, procesos, permisos, usuarios, redes,
- **sys** temas específicos al sistema operativo donde se está ejecutando: propiedades de los ejecutables, de la configuración del sistema,...
- **csv** para trabajar en un nivel básico con archivos de datos de tipo csv
- **math** todo lo relacionado con las matemáticas: funciones, constantes,...
- **glob**: para trabajar con comodines al listar ficheros.
- **datetime**: fechas y horas
- **statistics**: para realizar operaciones estadísticas.
- **zip**: para trabajar con ficheros comprimidos zip.
- **time**: medida del tiempo y esperas.
- **Multimedia**: para reproducir y generar ficheros multimedia
- **Protocolos de Internet**: para utilizar los protocolos más frecuentes de internet.
- **random**: números y elecciones aleatorios

Ya hemos trabajado algunos de ellos y ahora vamos a ver ejemplos de otros.

MÓDULO SYS

El módulo sys tiene las utilidades propias del sistema operativo donde estamos ejecutando el intérprete:

- *sys.argv*: argumentos con los que hemos ejecutado el programa
- *sys.executable*: ejecutable del intérprete actual.
- *sys.exit()*: sale del programa python actual.
- *sys.float_info*: información sobre las características del tipo decimal.
- *sys.getsizeof(objeto)*: nos dice la memoria que ocupa el objeto indicado.
- *sys.maxsize*: máximo entero que podemos usar.
- *sys.modules*: diccionario con los módulos cargados.
- *sys.version*: versión del intérprete que estamos usando.

MÓDULO MATH

En el módulo math como no podía ser de otra forma nos vamos a encontrar todo tipo de funciones matemáticas:

- Funciones trigonométricas, como *sin*, *cos*, *tan*,... y sus inversas *asin*, *acos*, *atan*, ...
- Raíces cuadradas con *sqrt*
- Exponenciales con *exp* y logaritmos con *log*.
- Constantes como *math.pi*, *math.e*, *math.inf* (infinito)

Como ejemplo vamos a ver las diferentes formas de redondear un número decimal a entero

Empezamos por la función *round()* que produce el resultado al que estamos acostumbrados:

- Los números con parte decimal igual o menor de 0.5 se redondean al entero anterior
- Si la parte decimal es mayor que 0.5 se redondea al entero siguiente:

```
>>> numeros = (0.5, 0.51, 1.6, 2.5, 2.51)
>>> print('Round')
>>> for numero in numeros:
...     print(f'{numero} -> {round(numero)}')
Round
0.5 -> 0
0.51 -> 1
1.6 -> 2
2.5 -> 2
2.51 -> 3
```

El módulo *math* incluye otras 2 formas de redondear:

- *math.floor()* que redondea siempre al entero anterior

```
>>> import math
>>> print('math.floor')
>>> for numero in numeros:
...     print(f'{numero} -> {math.floor(numero)}')
math.floor
0.5 -> 0
0.51 -> 0
1.6 -> 1
2.5 -> 2
2.51 -> 2
```

- *math.ceil()* redondea al entero superior

```
>>> import math
>>> print('math.ceil')
>>> for numero in numeros:
...     print(f'{numero} -> {math.ceil(numero)}')
math.ceil
0.5 -> 1
0.51 -> 1
1.6 -> 2
2.5 -> 3
2.51 -> 3
```

Además el módulo tiene otros métodos, como por ejemplo **gcd** que nos permite calcular el máximo común divisor:

```
>>> import math
>>> math.gcd(15,5)
5
```

Trabajando con fracciones

Aunque no forman parte del módulo math, sino que tienen su propio módulo, en la biblioteca estándar de Python podemos encontrar las clases necesarias para trabajar con *fracciones*. El módulo no podía tener otro nombre que **fractions**, donde trabajaremos con la clase **Fraction**, definiendo su *numerador* y su *divisor*:

```
>>> from fractions import Fraction
>>> f1 = Fraction(1,2)
>>> f2 = Fraction(1,3)
```

Por supuesto podemos imprimirlas:

```
>>> print(f1)
1/2
>>> print(f2)
1/3
```

Vemos que podemos operar entre ellas si recurrir a un valor decimal, lo que las hace más exactas:

```
>>> f3 = f1 + f2
>>> f3
Fraction(5, 6)
```

Y también podemos hacer operaciones entre fracciones y números "normales":

```
>>> f4 = 3 * f2
>>> f4
Fraction(1, 1)
>>> 1 == f4
True
```

y que en todo momento se simplifican los resultados:

```
>>> f5 = Fraction(3,24)
>>> f5
Fraction(1, 8)
```

ALEATORIEDAD CON RANDOM

Podemos usar el módulo random cuando necesitamos algo relacionado con lo aleatorio .

Tenemos varias posibilidades dependiendo de lo que necesitemos: * *randint(minimo,maximo)*: Nos da un número entero comprendido entre el valor mínimo y máximo incluyendo los dos extremos:

```
form random import randint
# Recordar que se incluye el máximo
# minimo <= randint(minimo,maximo) <= maximo
```

```
for i in range(10):
    print(randint(0,10))
```

- *choice(coleccion)* : Devuelve uno de los elementos de la colección que le hemos pasado:

```
from random import choice
colores = ('rojo', 'verde', 'azul', 'blanco', 'negro', 'amarillo')
for i in range(5):
    print (choice(colores))
```

- *sample(coleccion, N)*: Elige N elementos distintos de la colección que la hemos pasado y los devuelve como una tupla

```
from random import sample
colores = ('rojo', 'verde', 'azul', 'blanco', 'negro', 'amarillo')
print(sample(colores,4))
```

- *shuffle(lista)*: desordena aleatoriamente la lista que le pasamos.

Ejemplo: Generador de password

```
chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ.,-:;<>][=-_+'
longitud = int(input('¿longitud de la contraseña? '))
for i in range(longitud):
    passwd+=random.choice(chars)
    print(passwd)
```

A veces, en contra de lo que pudiera parecer, queremos que se reproduzca una misma secuencia de números aleatorios (para pruebas, para cierta predictibilidad,...). En ese caso podemos hacer usar el método *seed(semilla)* que hará que toda la secuencia se repita si usamos la misma *semilla*:

```
>>> from random import seed,randint
>>> seed(666)
>>> for i in range(5):
...     print(randint(0,10))
7
6
6
4
8
>>> seed(666)
>>> for i in range(5):
...     print(randint(0,10))
7
6
6
4
8
```

MIDIENDO TIEMPOS

El módulo *time* está pensado para medir tiempos.

```
>>> import time
>>> print(time.time()) # segundos desde 1/1/1970
1664454320.520648
>>> print(time.localtime()) # estructura con los datos de fecha y hora
time.struct_time(tm_year=2022, tm_mon=9, tm_mday=29, tm_hour=14, tm_min=25,
tm_sec=49, tm_wday=3, tm_yday=272, tm_isdst=1)
>>> print(time.strftime('%Y-%m-%d_%H:%M:%S',time.localtime()))
2022-09-29_14:28:06
```

- *time()*: Nos da un número decimal donde la parte entera son los segundos desde el 1 de enero de 1970 y la decimal la fracción de segundo. Este formato se conoce como **timestamp**
- *localtime()* Nos da una estructura con la fecha y hora descompuesta
- *strftime(formato)*: nos devuelve una cadena con el dato de fecha y hora con el formato que le indicamos
- *sleep(segundos)*: detiene la ejecución del programa durante los *segundos* indicados. Este valor puede ser decimal.

También podemos usar *time()* para ver el tiempo que ha pasado entre dos momentos:

```
>>> inicio = time.time()
>>> final = time.time()
>>> print(f'Han pasado {final-inicio} segundos')
Han pasado 8.950223922729492 segundos
```

Operaciones entre fechas con datetime

El módulo datetime está pensado para operar con fechas.

- *datetime(año,mes,dia, hora, minuto, segundo)* crea un objeto con fecha (podemos omitir los que valores de hora, minuto y segundo).
- datetime.year o datetime.month, ... nos da los valores esperados.
- *datetime.now()* nos da la fecha actual
- *datetime.strftime(formato)* convierte la fecha a una cadena con el formato indicado:

```
>>> from datetime import datetime
>>> datetime.now().strftime('%Y-%m-%d_%H:%M:%S')
'2021-03-25_20:52:38'
```

- Podemos convertir un valor de tipo *time* (timestamp) a datetime con:

```
>>> datetime.fromtimestamp(time.time())
datetime.datetime(2022, 9, 29, 14, 31, 43, 116377)
```

- Podemos restar fechas, obteniendo una estructura *timedelta* de resultado, de donde podemos extraer los días con *days*, los segundos con *seconds* y los microsegundos con *microseconds*. Código

```
>>> from datetime import datetime
>>> nacimiento = datetime(1970,12,25,15,0,0) # Fecha y hora de nacimiento
>>> hoy = datetime.now()
```

```
>>> diferencia = hoy - nacimiento
>>> print(f'han pasado {diferencia}')
han pasado 18905 days, 23:09:40.741071
>>> print(f'tienes {diferencia.days} días, y {diferencia.seconds} segundos')
tienes 18905 días, y 83380 segundos
>>> horas = diferencia.seconds // 3600 # segundos que tiene 1 hora
>>> restoSegundos = diferencia.seconds - horas * 3600
>>> minutos = restoSegundos // 60
>>> segundos = restoSegundos - minutos * 60
>>> print(f'o {diferencia.days} días, {horas} horas, {minutos} minutos y
{segundos} segundos')
o 18905 días, 23 horas, 9 minutos y 40 segundos
```

- Para obtener el día actual podemos usar *today()*

```
from datetime import date
hoy = date.today()
print(hoy)
# para formatearlo a Mes día, año
print(hoy.strftime('%B %d, %Y'))

# para saber el día de la semana
print(hoy.weekday())
```

Vemos que nos muestra el número de día de la semana.

También podemos trabajar con los nombres de los días de la semana o de los meses, para lo que usaremos el módulo **calendar**, que dispone de varias listas con nombres:

- **month_name** con los nombres de los meses
- **month_abbr** con las abreviaturas de los nombres de los meses
- **day_name** con los nombres de los días
- **day_abbr** con las abreviaturas de los nombres de los días

La lista de los meses tiene un elemento vacío en la primera posición (posición 0) para que se correspondan los nombres con los números habituales (enero - 1, ...), con lo que podemos sacar una lista de los nombres de los meses con:

```
>>> import calendar
>>> calendar.month_name[1:]
['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August',
'September', 'October', 'November', 'December']
```

O la de los días de la semana con:

```
>>> import calendar
>>> calendar.day_name[:]
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

También include constantes de los días de la semana y algunos meses (no sé porqué on están todos...) que corresponde con su número, por ejemplo *calendar.February* vale *2* o *calendar.TUESDAY* vale *1* y *calendar.MONDAY* vale *0*

Usando esto podemos recuperar el nombre del día de la semana:

```
# y para saber el nombre del día usamos calendar
>>> import calendar
>>> from datetime import date
>>> hoy = date.today()
>>> print(calendar.day_name[hoy.weekday()])
monday
```

Para obtener los nombres de los meses y días en el idioma local debemos establecer la localización con el módulo *locale*. Por ejemplo para establecer el "español" de "España" haríamos

```
>>> import locale
>>> locale.setlocale(locale.LC_ALL,'es_ES.UTF-8')
'es_ES.UTF-8'
```

A partir de este momento este intérprete mostrará los resultados traducidos:

```
>>> import calendar
>>> calendar.month_name[1:]
['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio', 'julio', 'agosto',
'septiembre', 'octubre', 'noviembre', 'diciembre']
>>> calendar.day_name[:]
['lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo']
```

Imprimiendo calendarios con *calendar*

También podemos usar *calendar*, como su nombre indica, para mostrar un calendario. Por ejemplo, el método *month(yy,mm)* devuelve una cadena con un calendario del mes mm del año yy:

```
>>> import calendar
>>> print(calendar.month(2004,5))
     mayo 2004
lu ma mi ju vi sá do
                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
```

También podemos mostrar el año completo con *calendar(year)*:

```
>>> import calendar
>>> print(calendar.calendar(2004))
                                  2004

      enero                    febrero                   marzo
lu ma mi ju vi sá do     lu ma mi ju vi sá do     lu ma mi ju vi sá do
          1  2  3  4                        1       1  2  3  4  5  6  7
 5  6  7  8  9 10 11      2  3  4  5  6  7  8       8  9 10 11 12 13 14
12 13 14 15 16 17 18      9 10 11 12 13 14 15      15 16 17 18 19 20 21
19 20 21 22 23 24 25     16 17 18 19 20 21 22      22 23 24 25 26 27 28
```

```
26 27 28 29 30 31         23 24 25 26 27 28 29      29 30 31

       abril                    mayo                     junio
lu ma mi ju vi sá do      lu ma mi ju vi sá do      lu ma mi ju vi sá do
          1  2  3  4                      1  2          1  2  3  4  5  6
 5  6  7  8  9 10 11       3  4  5  6  7  8  9       7  8  9 10 11 12 13
12 13 14 15 16 17 18      10 11 12 13 14 15 16      14 15 16 17 18 19 20
19 20 21 22 23 24 25      17 18 19 20 21 22 23      21 22 23 24 25 26 27
26 27 28 29 30            24 25 26 27 28 29 30      28 29 30
                          31

       julio                    agosto                   septiembre
lu ma mi ju vi sá do      lu ma mi ju vi sá do      lu ma mi ju vi sá do
          1  2  3  4                         1            1  2  3  4  5
 5  6  7  8  9 10 11       2  3  4  5  6  7  8       6  7  8  9 10 11 12
12 13 14 15 16 17 18       9 10 11 12 13 14 15      13 14 15 16 17 18 19
19 20 21 22 23 24 25      16 17 18 19 20 21 22      20 21 22 23 24 25 26
26 27 28 29 30 31         23 24 25 26 27 28 29      27 28 29 30
                          30 31

      octubre                  noviembre                 diciembre
lu ma mi ju vi sá do      lu ma mi ju vi sá do      lu ma mi ju vi sá do
             1  2  3       1  2  3  4  5  6  7            1  2  3  4  5
 4  5  6  7  8  9 10       8  9 10 11 12 13 14       6  7  8  9 10 11 12
11 12 13 14 15 16 17      15 16 17 18 19 20 21      13 14 15 16 17 18 19
18 19 20 21 22 23 24      22 23 24 25 26 27 28      20 21 22 23 24 25 26
25 26 27 28 29 30 31      29 30                     27 28 29 30 31
```

Recuperando fechas desde texto

A veces queremos recuperar una fecha de un texto, es decir, tenemos una cadena que sabemos que incluye información de fecha y de hora y queremos recuperar un datetime para poder utilizarlo de una forma más sencilla.

Para ello utilizaremos el método **parse** de la clase *datetime.parser* que nos permite recuperar tanto la fecha desde una cadena que sabemos que contiene una fecha, como desde un texto, por ejemplo una línea de un mensaje de error de un programa. A partir de ahí también podemos recuperar una fecha concreta.

Si tenemos seguridad de que la cadena incluye sólo una fecha lo usaremos así:

```
>>> from dateutil.parser import parse
>>> fechaStr = '10-5-2021 18:45:12'
>>> parse(fechaStr)
datetime.datetime(2021, 10, 5, 18, 45, 12)
```

En el caso de que la fecha esté incluida dentro de un texto podemos usar el parámetro *fuzzy=True* que buscará la parte de la fecha y nos la devolverá:

```
>>> from dateutil.parser import parse
>>> log_line = 'INFO 2014-07-03T23:27:51 supybot Shutdown complete.'
>>> parse(log_line, fuzzy=True)
datetime.datetime(2014, 7, 3, 23, 27, 51)
```

También podemos instalar otro paquete que nos permite parsear fechas, como es **dateparser**:

```
pip3 install dateparser
```

Usando su método parse podemos convertir un texto a fecha, obteniendo un *datetime*. También funciona con fechas escritas en diferentes idiomas:

```
>>> from dateparser import parse
>>> fecha = '18 de mayo de 2004'
>>> parse(fecha)
datetime.datetime(2004, 5, 18, 0, 0)
```