Unidad 2

Módulos de Utilidad

Manual del Estudiante

Maria del Carmen Sierra Fernández

Contenido

Nivel I: Programación con Python (cubre Examen 98-381)

Unidad 2: Módulos y Librerías Estándar

Módulos de Python

Al iniciar la terminal de Python tenemos disponibles de inmediato todos los tipos de datos y funciones que hemos visto hasta ahora, pero podemos tener funcionalidades adicionales importado módulos de Python, ya sean de la librería estándar o externas.

Para importar un módulo escribimos:

```
import <nombre del modulo>
```

Hay varias maneras de importar un módulo:

```
import math                 # importa el módulo math
import math as M            # importa el módulo math llamándolo M
from math import sin, cos   # importa las funciones sin, cos y pi
from math import *          # importa todas las funciones de math
```

Librerías estándar de Python

La instalación básica de Python viene con una muy completa librería de módulos para todo tipo de tareas, incluyendo acceso a ficheros y directorios, compresión de ficheros, ejecución recurrente (multihilo), email, html, xml, csv, entre muchas otras, disponible en la documentación de la librería estándar de Python.

Funciones del sistema (**sys**)

El módulo sys proporciona funciones y variables que se utilizan para manipular diferentes partes del entorno de ejecución de Python. El módulo sys se importa como:

```
import sys
```

Variables

`sys.maxsize:` Informa el tamaño del puntero de la plataforma y eso limita el tamaño de las estructuras de datos de Python, como cadenas y listas. A partir de la versión 3 de Python no está disponible la variable maxint que retornaba el número positivo entero mayor, soportado por Python.

```
sys.maxsize
9223372036854775807
```

`sys.path:` Es una lista de directorios/carpetas donde Python busca los módulos cuando realizamos un import. Ya que es una lista podemos agregar y quitar elementos.

```
sys.path
```

```
C:\\Users\\admin\\Desktop\\My Curso
Python\\EntornoVirtual\\Ejercicios\\Ejemplos Clase',
 'C:\\Users\\admin\\Anaconda3\\envs\\AAUT1\\python37.zip',
 'C:\\Users\\admin\\Anaconda3\\envs\\AAUT1\\DLLs',
 'C:\\Users\\admin\\Anaconda3\\envs\\AAUT1\\lib',
 'C:\\Users\\admin\\Anaconda3\\envs\\AAUT1',
 …
```

`sys.platform:` retorna la plataforma sobre la cual se está ejecutando el intérprete.

```
sys.platform
'win32'
```

`sys.executable:` retorna la ruta absoluta del ejecutable correspondiente al intérprete de Python.

```
sys.executable
'C:\\Users\\admin\\Anaconda3\\envs\\AAUT1\\python.exe'
```

`sys.version:` Es una cadena que nos indica la versión de intérprete que estamos utilizando.

```
print sys.version
'3.7.7 (default, May  6 2020, 11:45:54) [MSC v.1916 64 bit (AMD64)]'
```

`sys.argv:` Es la lista de argumentos con la cual se invocó al intérprete.

```
sys.argv
['C:\\Program Files\\JetBrains\\PyCharm Community Edition
2020.2\\plugins\\python-ce\\helpers\\pydev\\pydevconsole.py',
'--mode=client', '--port=58252']
```

Funciones

`sys.exit():` permite terminar la ejecución del script devolviendo un valor.

`sys.getdefaultencoding():` Retorna la codificación de caracteres por defecto

```
sys.getdefaultencoding()
'utf-8'
```

`sys.getfilesystemencoding():` Retorna la codificación de caracteres que se utiliza para convertir los nombres de archivos Unicode en nombres de archivos del sistema.

```
sys.getfilesystemencoding()
'utf-8'
```

Creación y gestión de ficheros (Módulo **os**)

La forma más directa y práctica de interactuar con el sistema, independientemente de la plataforma, es empleando el módulo *os*, que básicamente es una interfaz con el sistema operativo del ordenador que ejecuta el programa.

Luego de importar la librería, podemos utilizar las funciones y métodos disponibles:

```
import os # La importación del módulo, al principio del programa
```

- **os.name:** retorna el nombre del sistema operativo (posix, nt, os2, ce, java y riscos).

```
print(os.name)
nt
```

- **os.getcwd():** esta función retorna el directorio actual de trabajo del archivo utilizado para ejecutar el código.

```
path = os.getcwd()
print(path)
C:\Users\admin\Desktop\My Curso Python
```

- **os.chdir(*path*):** Cambia el directorio de trabajo actual a *path*.

```
PATH_FILES = r"C:\Users\admin\Projects\BigDataProject\"
os.chdir(PATH_FILES)
```

- **os.path.isdir(path):** Esta función específica si la ruta de acceso a un directorio existe o no.

```
PATH_IMAGES = PATH_FILE + "\imagen"

if os.path.isdir(PATH_IMAGES):  # Retorna True o False
    os.chdir(PATH_IMAGES)
else:
    print("Directorio o ruta no existe\n")

print("La ruta actual es: ", os.getcwd())
```

- **os.mkdir(path):** Esta función crea un directorio *path*

```
PATH_IMAGES = PATH_FILE + "\imagenes"

if os.path.isdir(PATH_IMAGES):
    os.chdir(PATH_IMAGES)
else:
    print("Directorio o ruta no existe\n")
    os.mkdir('imagenes') # Crear la carpeta imágenes
    os.chdir(PATH_IMAGES)

print("La ruta actual es: ", os.getcwd())
```

Trabajando con fechas y tiempo (Módulo **datetime**)

La librería estándar de Python incluye varios módulos para tratar y manipular fechas, tiempo e intervalos.

El módulo principal es `datetime`, que permite trabajar con fechas y tiempo mientras que el módulo `time`, ofrece métodos avanzados para tiempo, ignorando la fecha y `calendar` funciones generales relacionadas a calendarios.

Una vez importamos la librería `from datetime import <modulo>`, tendremos acceso a los métodos de los módulos.

```
from datetime import timedelta
from datetime import date
from datetime import datetime
```

Objeto timedelta

El objeto timedelta, es la base para operaciones con fechas. Representa una duración de tiempo, es decir, la diferencia entre dos fechas o instantes de tiempo.

```
datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0,
minutes=0, hours=0, weeks=0)
```

Almacena internamente sólo *los días, segundos y microsegundos*, por ende, los argumentos se convierten en esas unidades:

- Un milisegundo se convierte en 1000 microsegundos.
- Un minuto se convierte en 60 segundos.
- Una hora se convierte en 3600 segundos.
- Una semana se convierte en 7 días.

Ejemplo:

```
from datetime import timedelta

delta = timedelta(
     days=50,
     seconds=27,
     microseconds=10,
     milliseconds=29000, # convierte a microseg (29000*1000)
     minutes=5,          # convierte los min a segundos (5*60)
     hours=8,            # convierte horas a segundos (8*3600)
     weeks=2             # convierte semanas a días (2*7)
)

delta
datetime.timedelta(days=64, seconds=29156, microseconds=10)
```

Operaciones con fechas

Cuando necesitemos realizar operaciones con fechas/tiempo, podemos utilizar un objeto `timedelta` como referencia.

Por ejemplo: Crear un objeto que represente un (1) día

```
timedelta(days=1)   # representa 1 día
datetime.timedelta(days=1)

print(timedelta(days=1))
1 day, 0:00:00
```

Restar días:

```
from datetime import timedelta
f_actual = datetime.utcnow()
print("Hoy es: ", f_actual)

ayer = f_actual - timedelta(days=1)
print("Ayer fue:", ayer)

Hoy es: 2020-09-11 14:53:41.004725
Ayer fue: 2020-09-10 14:53:41.004725
```

Restar horas:

```
print("Ahora: ", f_actual)
hace_una_hora = f_actual - timedelta(hours=1)
print("Hace una hora: ", hace_una_hora)

Ahora: 2020-09-11 14:57:12.396652
Hace una hora: 2020-09-11 13:57:12.396652
```

Sumar días:

```
print("Hoy es: ", f_actual)
manana = f_actual + timedelta(days=1)
print("Mañana será: ", manana)

Hoy es: 2020-09-11 14:57:12.396652
Mañana será: 2020-09-12 14:57:12.396652
```

Parámetros que se pueden pasar a timedelta

Además de días y horas, un objeto timedelta se puede inicializar con los siguientes valores: weeks, days, hours, minutes, seconds, milliseconds y microseconds.

```
timedelta(weeks=1)
datetime.timedelta(days=7)

print(timedelta(weeks=1))
7 days, 0:00:00
```

Objeto date

Representa una fecha (año, mes y día) en un calendario idealizado, el calendario gregoriano actual se extiende indefinidamente en ambas direcciones.

```
from datetime import date
```

El método `date.today()` retorna la fecha local actual

```
today = date.today()  # Obtenemos el día actual
print("Día actual: ", today)
Día actual:  2020-09-09
```

Atributos:

```
print("El día actual es {}".format(today.day))
print("El mes actual es {}".format(today.month))
print("El año actual es {}".format(today.year))

El día actual es 9
El mes actual es 9
El año actual es 2020
```

Objeto datetime

El objeto datetime es un único objeto que contiene toda la información de un objeto *date* y un objeto *time*.

Como un objeto date, datetime asume el calendario gregoriano actual extendido en ambas direcciones; como un objeto time, datetime supone que hay exactamente 3600*24 segundos en cada día.

```
datetime.datetime(year, month, day, hour=0, minute=0, second=0,
microsecond=0, tzinfo=None, *, fold=0)

# Los argumentos son: Año, Mes, Día, Hora, Minutos, Segundos,
Milisegundos.
```

```
from datetime import datetime

new_date = datetime(2019, 2, 28, 10, 15, 00, 00000)
print(new_date)
new_date

2019-02-28 10:15:00
datetime.datetime(2019, 2, 28, 10, 15)
```

Métodos:

- `datetime.today()`: Retorna la fecha y hora local actual, con *tzinfo* None.

```
mi_fecha = datetime.today()
print("Fecha actual: ", mi_fecha)
Fecha actual: 2020-09-09 21:08:20.012032
```

- `datetime.now(tz=None)`: Retorna la fecha y hora local actual. Si *tz* no es None, debe ser una instancia de una subclase *tzinfo*, y la fecha y hora actuales se convierten en la zona horaria de *tz*.

```
fecha = datetime.now()
print("Fecha actual: ", fecha)
Fecha actual: 2020-09-09 21:08:20.013029
```

- `datetime.utcnow()`: Retorna la fecha y hora UTC (Current time in Coordinated Universal Time (UTC)) actual, con *tzinfo* None.

```
fechautc = datetime.utcnow()
print("Fecha actual: ", fechautc)
Fecha actual: 2020-09-09 19:08:20.013029
```

Atributos:

- datetime.year
- datetime.month
- datetime.day
- datetime.hour
- datetime.minute
- datetime.second
- datetime.microsecond

Ejemplos:

```
now = datetime.today()

print("El día actual es {}".format(now.day))
print("El mes actual es {}".format(now.month))
print("El año actual es {}".format(now.year))

print("La hora actual es {}".format(now.hour))
print("Los minutos actuales son {}".format(now.minute))
print("Los segundos actuales son {}".format(now.second))
print("Los microsegundos actuales son
{}".format(now.microsecond))

El día actual es 11
El mes actual es 9
El año actual es 2020
La hora actual es 16
Los minutos actuales son 12
Los segundos actuales son 52
```

```
Los microsegundos actuales son 49723
```

Métodos

timetuple(): Retorna una estructura (time.struct_time)

```
time.struct_time((d.year, d.month, d.day,
                  d.hour, d.minute, d.second,
                  d.weekday(), yday, dst))
```

donde:

```
yday = d.toordinal() - date(d.year, 1, 1).toordinal() + 1
```

es el número de día dentro del año actual que comienza con 1 para el 1 de enero.

Ejemplo:

```
from datetime import datetime

fecha_actual = datetime.today()
attributesInTuple = fecha_actual.timetuple()

attributesInTuple # Mostrar la estructura

    time.struct_time(tm_year=2020, tm_mon=9, tm_mday=9,
    tm_hour=21, tm_min=52, tm_sec=28, tm_wday=2, tm_yday=253,
    tm_isdst=-1)

for attribute in attributesInTuple: # Muestra los atributos
    print(attribute)

2020
9
9
21
52
28
2
253
-1
```

Método strptime(): nos permite convertir una fecha en formato cadena, a un objeto datetime.

Sintaxis: `strptime(date_string, format)`

Ejemplo:

```
fecha_cadena = '10/9/20'
print(type(fecha_cadena))
<class 'str'>
```

Si queremos cambiar la fecha_cadena de '10/9/20' a un objeto datetime:

```
fecha = datetime.strptime(fecha_cadena, '%m/%d/%y')
fecha
datetime.datetime(2020, 10, 9, 0, 0)
```

Método `strftime()`: El objeto datetime tiene un método para formatear objetos de fecha en cadenas legibles. **strftime()** toma un parámetro, formato, para especificar el formato de la cadena que retornará:

Sintaxis: `strftime(format)`

Ejemplo:

1. Si necesitamos mostrar el nombre del mes:

```
from datetime import datetime
mes = datetime(2019, 3, 1)
print(mes.strftime("%B"))
March
```

2. Si necesitamos mostrar el año:

```
fecha = datetime.today()
datetime.strftime(fecha,'%Y')
‘2020’
```

La siguiente tabla, muestra una referencia de los formatos disponibles de fecha:

Directiva	Descripción	Ejemplo
%a	Día de la semana, versión corta.	Fri
%A	Día de la semana, versión completa.	Friday
%w	Día de la semana como número 0-1, 0 es domingo.	5
%d	Día del mes 01-31.	11
%b	Nombre del mes, versión corta.	Sep
%B	Nombre del mes, versión completa	September
%m	Mes como número 01-12	09
%y	Año, versión corta, sin siglo.	20

%Y	Año, versión completa	2020
%H	Hora 00-23	18
%I	Hora 00-12	06
%p	AM/PM	PM
%M	Minuto 00-59	41
%S	Segundo 00-59	12
%f	Microsegundo 000000-999999	548513
%z	UTC offset	100
%Z	Zona horaria	CST
%j	Día del año 001-366	255
%U	Número de semana del año, domingo como primer día de la semana, 00-53	52
%W	Número de semana del año, lunes como primer día de la semana, 00-53lunes	52
%c	Representación de fecha y hora apropiada de la configuración regional.	Fri Sep 11 18:18:55 2020
%x	Representación de fecha apropiada de la configuración regional.	12/31/19
%X	Representación de la hora apropiada de la configuración regional.	5:30:00
%%	A % carácter	%

Ejemplos:

```
# Mostrar la fecha actual con y sin formato

      from datetime import datetime
      print("La fecha y hora actual: " ,
            datetime.today()) # Devuelve un objeto datetime
      print("La fecha y hora con formato: " ,
            datetime.today().strftime("%Y-%m-%d %H:%M"))

      La fecha y hora actual: 2020-09-13 20:51:11.582503
      La fecha y hora con formato: 2020-09-13 20:51

# Mostrar la fecha actual con formato 13 de September del 2020

      fecha = datetime.now()
      print("{} de {} del {}".format(fecha.strftime('%d'),
                                     fecha.strftime('%B'),
                                     fecha.strftime('%Y')))
     13 de September del 2020

# Mostrar el mes en español
```

```
fecha = datetime.now()
meses = ("Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
         "Julio", "Agosto", "Septiembre", "Octubre",
         "Noviembre", "Diciembre")
dia = fecha.day
mes = meses[fecha.month - 1]
año = fecha.year
print("{} de {} del {}".format(dia, mes, año))
13 de Septiembre del 2020
```

Las siguientes sentencias utilizan los diferentes formatos. Crea un programa Python, cópialas y revisa los resultados obtenidos:

```
from datetime import datetime, date

print("Día de la semana: ",  datetime.today().strftime("%a"))
print("Día de la semana: ",  datetime.today().strftime("%A"))
print("Día de la semana: ",  datetime.today().strftime("%w"))

print("Día del mes: ",  datetime.today().strftime("%d"))

print("Mes del año: ", datetime.today().strftime("%b"))
print("Mes del año: ", datetime.today().strftime("%B"))
print("Mes del año: ", datetime.today().strftime("%m"))

print("Año actual: ", datetime.today().strftime("%y"))
print("Año actual: ", datetime.today().strftime("%Y"))

print("Hora 00-23: ", datetime.today().strftime("%H"))
print("Hora 00-12: ", datetime.today().strftime("%I"))
print("AM/PM: ", datetime.today().strftime("%p"))
print("Minutos: ", datetime.today().strftime("%M"))
print("Segundos: ", datetime.today().strftime("%S"))
print("Microsegundos: ", datetime.today().strftime("%f"))
print("Día del año: ",  datetime.today().strftime("%j"))

print("Semana del año (Domingo primer día): ",
      datetime.today().strftime("%U"))
print("Semana del año (Lunes primer día): ",
      datetime.today().strftime("%W"))

print("Fecha y hora (Configuración regional): ",
      datetime.today().strftime("%c"))
print("Fecha (Configuración regional): ",
      datetime.today().strftime("%x"))
print("Hora (Configuración regional): ",
      datetime.today().strftime("%X"))
```

```
print("Año actual: ", date.today().strftime("%Y"))
print("Año actual: ", date.today().strftime("%y"))
print("Mes del año: ", date.today().strftime("%B"))
```

Otros Métodos

```
fecha = datetime.now(timezone.utc)
```

`fecha.date()`: retorna el objeto date

`fecha.time()` y `fecha.timetz()`: retorna el objeto time sin y con tzinfo
`fecha.tznam()`: retorna la tzinfo
`fecha.weekday()`: Retorna el día de la semana, lunes es 0 y domingo 6
`fecha.isoweekday()`: Retorna el día de la semana, lunes es 1 y domingo 7

Ejemplo:

```
fecha = datetime.now(timezone.utc)

print("Fecha timezone.utc", fecha)
print("Hora con zona horaria:" , fecha.timetz())
print("Nombre de la zona horaria:" ,fecha.tzname())
```

```
Fecha timezone.utc 2020-09-13 21:11:49.626061+00:00
Time zone: 21:11:49.626061+00:00
Time zone name: UTC
```

La biblioteca estándar de Python no contiene información de la zona horaria, necesitamos un módulo de terceros para esto; la elección habitual es pytz

```
from datetime import datetime
from pytz import timezone

zonas = ['UTC', 'US/Pacific', 'Europe/Berlin',
         'Australia/Adelaide',
         'America/New_York']
for zona in zonas:
    fecha = datetime.now(timezone(zona))
    print(fecha.strftime("%Y-%m-%d %H:%M:%S %Z%z"),zona)
```

```
2020-09-13 21:46:05 UTC+0000 UTC
2020-09-13 14:46:05 PDT-0700 US/Pacific
2020-09-13 23:46:05 CEST+0200 Europe/Berlin
2020-09-14 07:16:05 ACST+0930 Australia/Adelaide
2020-09-13 17:46:05 EDT-0400 America/New_York
```

Objeto time

Un objeto time representa a una hora del día (local), independiente de cualquier día en particular, y está sujeto a ajustes a través de un objeto tzinfo (información sobre una zona horaria particular)

```
datetime.time(hour=0, minute=0, second=0, microsecond=0,
              tzinfo=None, *, fold=0)
```

Método `time()`: retorna la hora como un número con punto flotante expresado en segundos (timestamp).

```
import time
time.time()

1600037302.6448398
```

`time.localtime(secs)`: retorna la hora local como una estructura

```
time.localtime( time.time() )
time.struct_time(tm_year=2020, tm_mon=9, tm_mday=14, tm_hour=0,
tm_min=48, tm_sec=22, tm_wday=0, tm_yday=258, tm_isdst=1)
```

`time.asctime(t)`: Convierte una tupla o estructura de tiempo a un string.

```
print(time.asctime( time.localtime(time.time()) ))
Mon Sep 14 00:48:22 2020
```

`time.sleep(secs)`: Suspende la ejecución del subproceso actual durante un número de segundos

```
import time

print("Esto se imprime de inmediato.")
time.sleep(5.4)
print("Esto se imprime a  5.4 segundos.")

Esto se imprime de inmediato.
Esto se imprime a 5.4 segundos.
```

Otros métodos (gmtime, strftime, strptime)

```
import time
from time import gmtime, strftime, strptime

strftime("%a, %d %b %Y %H:%M:%S +0000", gmtime())
'Sun, 13 Sep 2020 23:03:50 +0000'

time.strptime("30 Nov 00", "%d %b %y")
```

```
time.struct_time(tm_year=2000, tm_mon=11, tm_mday=30,
tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=335,
tm_isdst=-1)
```

`gmtime(secs)`: Convierte el tiempo expresado en segundos (timestamp) a una estructura de tiempo.

```
hora = gmtime()
time.struct_time(tm_year=2020, tm_mon=9, tm_mday=13, tm_hour=23,
tm_min=3, tm_sec=50, tm_wday=6, tm_yday=257, tm_isdst=0)

hora.tm_hour
23
hora.tm_min
3
```

Formato de fecha ISO

El formato internacional definido por ISO (IS0 8601) define un sistema numérico de fechas como se muestra a continuación: AAAA-MM-DD o la representación completa: YYYY-MM-DDTHH:MM:SS (2007-11-03T24:12:03)

Donde:

- AAAA es el año [todos los dígitos, p.ej. 2012]
- MM es el mes [01 (Enero) hasta 12 (Diciembre)]
- DD es el día [01 hasta 31]
- HH horas [00 hasta 23]
- MM minutos [00 hasta 59]
- SS segundos [00 hasta 59]

Por ejemplo, "05 de Abril de 2020", en este formato internacional se escribe: 2020-04-05.

```
# Local to ISO 8601:
datetime.now().isoformat()
2020-09-14T00:14:01.309464

# UTC to ISO 8601:
datetime.utcnow().isoformat()
2020-09-14T00:14:01.309464

datetime.now().replace(microsecond=0).isoformat()
2020-09-14T00:14:01

# UTC to ISO 8601 with TimeZone information (Python 3):

datetime(2019, 5, 18, 15, 17,tzinfo=timezone.utc).isoformat()
2019-05-18T15:17:00+00:00
```

```
datetime.now().astimezone().isoformat()
2020-09-14T00:14:01.309464+02:00
```

Formato de Fecha timestamp

Es bastante común almacenar la fecha y la hora como una marca de tiempo en una base de datos. Una marca de tiempo Unix es el número de segundos entre una fecha determinada y el 1 de enero de 1970.

Convertir un timestamp a datetime

```
from datetime import datetime

timestamp = 1545730073
fecha = datetime.fromtimestamp(timestamp)

print("Timestamp:   ", timestamp)
print("Fecha:       ", fecha)
print("type(fecha): ", type(fecha))
Timestamp:    1545730073
Fecha:        2018-12-25 10:27:53
type(fecha):  <class 'datetime.datetime'>
```

Convertir un datetime a timestamp

```
fecha = datetime.now()

timestamp = datetime.timestamp(fecha)

print("Fecha", fecha)
print("Timestamp =", timestamp)
Fecha 2020-09-14 00:36:11.872556
Timestamp = 1600036571.872556
```

Módulo Calendar

Este módulo te permite generar calendarios como el programa Unix cal, y proporciona funciones útiles adicionales relacionadas con el calendario. Por defecto, estos calendarios tienen el lunes como el primer día de la semana, y el domingo como el último (la convención europea). Use `setfirstweekday()` para establecer el primer día de la semana en domingo (6) o en cualquier otro día de la semana.

Establece el día de la semana (0 lunes y 6 domingo) para empezar cada semana, también puede utilizar los valores MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY y SUNDAY.

Por ejemplo, para fijar el primer día de la semana en domingo:

```
import calendar
calendar.setfirstweekday(calendar.SUNDAY)

calendar.setfirstweekday(6)
```

Un objeto Calendar proporciona varios métodos que se pueden utilizar para preparar los datos del calendario para dar formato

`calendar.calendar(año,w=2,l=1,c=6):` Devuelve una cadena multilínea con un calendario por año con un formato de 3 columnas separadas por los espacios de c. w es el equivalente al ancho entre caracteres de cada fecha;

```
calendar.calendar(2020)
```

`calendar.firstweekday()`:Nos devuelve la configuración actual para el primero día de la semana.

```
calendar.firstweekday()
0
```

`calendar.isleap(año):` Devuelve True si el año a evaluar es bisiesto, mientras que si es lo contrario devuelve False.

```
calendar.isleap(2020)
True
```

`TextCalendar(firstweekday=0):` Esta clase puede ser usada para generar calendarios de texto simple.

- `formatmonth(theyear, themonth, w=0, l=0):` Retorna el calendario de un mes en una cadena de varias líneas. Si se proporciona w, especifica el ancho de las columnas de fecha, que están centradas. Si se proporciona l, especifica el número de líneas que se utilizarán cada semana.
- `prmonth(theyear, themonth, w=0, l=0):` Imprime el calendario de un mes como lo retorna formatmonth().
- `formatyear(theyear, w=2, l=1, c=6, m=3):` Retorna un calendario de m columnas para todo un año como una cadena de varias líneas. Los parámetros opcionales w, l y c son para el ancho de la columna de la fecha, las líneas por semana y el número de espacios entre las columnas del mes, respectivamente.
- `pryear(theyear, w=2, l=1, c=6, m=3):` Imprime el calendario de un año entero como lo retorna formatyear().

Ejemplo:

```
cl = calendar.TextCalendar(calendar.SUNDAY)
```

```
calendario_sep = cl.formatmonth(2020, 9)
print(calendario_sep)
```

```
   September 2020
Su Mo Tu We Th Fr Sa
       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
```

El módulo calendar, también nos da la posibilidad de generar calendarios en formato HTML. Esto lo podemos lograr usando la clase HTMLCalendar.

```
class calendar.HTMLCalendar(firstweekday=0)
```

Esta clase puede utilizarse para generar calendarios HTML. Las instancias de HTMLCalendar tienen los siguientes métodos:

`formatmonth(theyear, themonth, withyear=True)`: Retorna el calendario de un mes como una tabla HTML. Si withyear es verdadero, el año será incluido en el encabezado, de lo contrario sólo se usará el nombre del mes.

`formatyear(theyear, width=3)`: Retorna el calendario de un año como una tabla HTML. width (por defecto a 3) especifica el número de meses por fila.

`formatyearpage(theyear, width=3, css='calendar.css', encoding=None)`: Retorna el calendario de un año como una página HTML completa. width (por defecto a 3) especifica el número de meses por fila. css es el nombre de la hoja de estilo en cascada que se debe usar. None puede ser pasada si no se debe usar una hoja de estilo. encoding especifica la codificación a ser usada para la salida (por defecto a la codificación por defecto del sistema).

Ejemplo:

```
import calendar

myCal = calendar.HTMLCalendar(calendar.SUNDAY)
print(myCal.formatmonth(2020, 10))
```

```
<table border="0" cellpadding="0" cellspacing="0" class="month">
<tr><th colspan="7" class="month">October 2020</th></tr>
<tr><th class="sun">Sun</th><th class="mon">Mon</th><th
class="tue">Tue</th><th class="wed">Wed</th><th class="thu">Thu</th><th
class="fri">Fri</th><th class="sat">Sat</th></tr>
...
```

October 2020						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
				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

Cómo iterar o recorrer un calendario

`iterweekdays()`: Retorna un iterador para los números del día de la semana que se usará durante una semana. El primer valor del iterador será el mismo que el valor de la propiedad firstweekday.

```
c = calendar.TextCalendar()
for dia_semana in c.iterweekdays():
    print(dia_semana)
0 1 2 3 4 5 6
```

`itermonthdates(year, month)`: Retorna un iterador para el mes month (1–12) en el año year. Este iterador retornará todos los días (como objetos datetime.date) para el mes y todos los días antes del inicio del mes o después del final del mes que se requieren para obtener una semana completa.

```
for fecha in c.itermonthdates(2021,4):
    print(fecha)

2021-03-29
2021-03-30
2021-03-31
2021-04-01
2021-04-02
2021-04-03
2021-04-04
…
```

`itermonthdays(year, month)`: Retorna un iterador para el mes month en el año year similar a itermonthdates(), pero no restringido por el intervalo datetime.date. Los días retornados serán simplemente el día de los números del mes. Para los días fuera del mes especificado, el número de día es 0.

```
for dia in c.itermonthdays(2021,4):
    print(dia)

0 0 0 1 2 3 4 5
```

`monthdays2calendar()`: retorna una matriz con tuplas que contienen el número de día del mes y el número de día de la semana (0-lunes, 1-martes, 2-miércoles ...)

```
for dia in c.itermonthdays2(2021,4):
    print(dia)
(0, 0)
(0, 1)
(0, 2)
(1, 3)
(2, 4)
(3, 5)
…
```

Servicios de internacionalización (Módulo **locale**)

El módulo locale abre el acceso a la base de datos y la funcionalidad de POSIX locale (acrónimo de Portable Operating System Interface, y X viene de UNIX como seña de identidad de la API) . El mecanismo POSIX locale permite a los programadores tratar ciertos problemas culturales en una aplicación, sin requerir que el programador conozca todos los detalles de cada país donde se ejecuta el software.

`locale.setlocale(category, locale=None)`
Si locale está dado y no es None, setlocale() modifica la configuración de localización para category. Las categorías disponibles son:

- locale.LC_ALL todas las categorías.
- locale.LC_NUMERIC configuraciones de números.
- locale.LC_TIME configuraciones de fechas.
- locale.LC_MONETARY configuraciones de moneda.
- locale.LC_COLLATE configuraciones de comparación de textos.
- locale.LC_CTYPE configuraciones de caracteres.

```
import locale

locale.setlocale(locale.LC_ALL, '')
'Spanish_Spain.1252'

locale.currency(1234.76)
'1234,76 €'
```

Ahora cambiemos toda la configuración a Reino Unido:

```
locale.setlocale(locale.LC_ALL, 'en-GB') #Reino Unido
locale.currency(1234.76)
'£1234.76'
```

Esto establece la configuración regional de todas las categorías en la configuración predeterminada del usuario (normalmente especificada en la variable de entorno LANG).

`locale.localeconv()`: retorna la base de datos de las convenciones locales como diccionario.

```
locale.localeconv()
{'int_curr_symbol': 'EUR',
 'currency_symbol': '€',
 'mon_decimal_point': ',',
 'mon_thousands_sep': '.',
 'mon_grouping': [3, 0],
 'positive_sign': '',
 'negative_sign': '-',
 'int_frac_digits': 2,
 'frac_digits': 2,
 'p_cs_precedes': 0,
 'p_sep_by_space': 1,
 'n_cs_precedes': 0,
 'n_sep_by_space': 1,
 'p_sign_posn': 1,
 'n_sign_posn': 1,
 'decimal_point': ',',
 'thousands_sep': '.',
 'grouping': [3, 0]}
```

Grouping: Secuencia de números que especifica qué posiciones relativas se espera el 'miles_sep'.

Funciones para obtener el idioma y la codificación.

Las siguientes funciones permiten obtener el idioma y la codificación de las cadenas de texto de distintos modos:

`locale.getdefaultlocale([envvars])`: determinar la configuración regional por defecto.

```
locale.getdefaultlocale()
('es_ES', 'cp1252')
```

`locale.getlocale(category=LC_CTYPE)`: retorna la configuración actual para la categoría de configuración regional dada como secuencia que contiene language code, coding. category puede ser uno de los valores LC_ * excepto LC_ALL. Por defecto es LC_CTYPE.

```
locale.getlocale(locale.LC_NUMERIC)
('es_ES', 'cp1252')

locale.getlocale(locale.LC_CTYPE)
('es_ES', 'cp1252')
```

`getpreferredencoding(do_setlocale=True)`: retorna la codificación utilizada para los datos de texto, según las preferencias del usuario.

```
locale.getpreferredencoding()
'cp1252'
```

`normalize(localename)`: retorna un código de configuración regional normalizado para el nombre configuración regional dado.

```
locale.normalize('es_ES')
'es_ES.ISO8859-1'
```

`resetlocale(category=LC_ALL)`: Establece la configuración regional de category a la configuración predeterminada.

```
locale.resetlocale(locale.LC_NUMERIC)
```

Funciones para dar formato a números

`Locale.currency(val, symbol=True, grouping=False, international=False)`: Formatea un número val según la configuración actual LC_MONETARY .

La cadena de caracteres retornada incluye el símbolo de moneda si symbol es verdadero, que es el valor predeterminado. Si grouping es verdadero, la agrupación se realiza con el valor. Si international es verdadero, se utiliza el símbolo de moneda internacional.

```
import locale

valor = 123456.785

locale.currency(valor, symbol=True, grouping=False)
'123456,79 €'

locale.currency(valor, symbol=False, grouping=False)
'123456,79'

locale.currency(valor, symbol=True, grouping=True)
'123.456,79 €'
```

El siguiente ejemplo, nos muestra la configuración de la Moneda de varios países:

```
import locale

conf_pais = [ ('USA',      'en_US'),
              ('France',   'fr_FR'),
              ('Spain',    'es_ES'),
              ('Portugal', 'pt_PT'),
              ('Poland',   'pl_PL'), ]
```

```
for name, loc in conf_pais:
    locale.setlocale(locale.LC_ALL, loc)
    print('%10s: %10s  %10s' % (name, locale.currency(1234.56),
                                locale.currency(-1234.56)))
```

```
      USA:   $1234.56  ($1234.56)
   France:  1234,56 €  -1234,56 €
    Spain:  1234,56 €  -1234,56 €
 Portugal:  1234,56 €  -1234,56 €
   Poland: 1234,56 zł  -1234,56 zł
```

`locale.format_string(format, val, grouping=False, monetary=False)`:
Formatea un número val según la configuración actual LC_NUMERIC . El formato sigue las convenciones del operador %.

```
import locale
valor = 123456.785

locale.format_string('%10.2f', valor, grouping=True)
'123,456.785'
locale.format_string('%10.2f', valor, grouping=False)
' 123456.79'
```

El parámetro grouping decide si debe imprimirse el separador de miles, y el parámetro monetary determina si se debe redondear a dos decimales.

El siguiente ejemplo, nos muestra la configuración numérica de varios países:

```
import locale

conf_pais = [  ('USA',      'en_US'),
               ('France',   'fr_FR'),
               ('Spain',    'es_ES'),
               ('Portugal', 'pt_PT'),
               ('Poland',   'pl_PL'),
            ]

print('%20s %15s %20s' % ('Locale', 'Integer', 'Float'))
for name, loc in conf_pais:
    locale.setlocale(locale.LC_ALL, loc)

    print('%20s' % name, locale.format_string('%15d', 123456,
                                              grouping=True),
                         locale.format_string('%20.2f', 123456.78,
                                              grouping=True))
```

```
              Locale         Integer               Float
                 USA         123,456          123,456.78
              France         123 456          123 456,78
               Spain         123.456          123.456,78
            Portugal         123 456          123 456,78
```

```
Poland          123 456          123 456,78
```

`locale.str(float)`: Formatea un número de punto flotante usando el mismo formato que la función integrada str(float), pero toma en cuenta el punto decimal.

```
locale.str(valor)
'123456,785'
```

Comparación de cadenas.

Para comparar cadenas se utiliza la función `strcoll(primera, segunda)`. Esta función retorna:

- un valor negativo si la primera cadena es menor que la segunda,
- un valor cero si son iguales,
- y un valor positivo si la primera cadena es mayor que la segunda.

```
locale.strcoll('Florero', 'Florista')
-1
locale.strcoll('Floricultura', 'Florecer')
1
locale.strcoll('Floral', 'Floral')
0
```

Cuando comparamos cadenas utilizando los operadores relacionales, si la cadena posee caracteres específicos de un idioma, el resultado puede no ser correcto:

```
"año"  > "aquello"
True
```

Al comparar las cadenas 'año' y 'aquello' utilizando los operadores relacionales, vemos que el resultado no es el esperado, porque en español, la ñ es alfabéticamente inferior a la q. Esto ocurre, porque hace una comparación por los códigos ASCII, pero si aplicamos la comparación localizada obtendremos el resultado correcto:

```
locale.strcoll( "año" , "aquello")
-1
```

Si queremos ordenar una lista de cadena teniendo en cuenta la internacionalización debemos aplicar una clave al método sort():

```
cadenas = ['Florero', 'Florista', 'Floricultura', 'Florecer',
           'Floral']
cadenas.sort(key=locale.strxfrm)
print(cadenas)

['Floral', 'Florecer', 'Florero', 'Floricultura', 'Florista']
```

`locale.strxfrm(string)`: Transforma una cadena en una que se puede utilizar en comparaciones de localización.

Generar números pseudo aleatorios (Módulo **random**)

Este módulo implementa generadores de números pseudo aleatorios para varias distribuciones.

Para los enteros, existe una selección uniforme dentro de un rango. Para las secuencias, existe una selección uniforme de un elemento aleatorio, una función para generar una permutación aleatoria de una lista in-situ y una función para el muestreo aleatorio sin reemplazo.

Para números reales, existen funciones para calcular distribuciones uniformes, normales (Gaussianas), log-normales, exponenciales negativas, gamma y beta.

```
import random

print(random.random())    # Random float:  0.0 <= x < 1.0

0.7506846663547462
```

Bookkeeping functions

`random.seed(x):` Inicializa el generador de números aleatorios. El generador de números aleatorios necesita un número para empezar (un valor de semilla), para poder generar un número aleatorio.

Donde x: es la semilla para el siguiente número aleatorio. Si se omite, el sistema necesita tiempo para generar el siguiente número aleatorio.

```
random.seed(42)
print(random.random())
0.6394267984578837

random.seed(42)
print(random.random())
0.6394267984578837

random.seed(3)
print(random.random())
0.23796462709189137
```

`random.getstate()`: Retorna un objeto capturando el estado interno del generador. Este objeto puede pasarse a setstate() para restaurar su estado.

`random.setstate(state)`: El state debería haberse obtenido de una llamada previa a getstate(), y setstate() reestablece el estado interno del generador al que tenía cuando se llamó a la función getstate().

Ejemplo:

```
print(random.random())    # imprimir un número random
```

```
state = random.getstate() # captura el state

print(random.random())    # imprime otro número random

random.setstate(state)    # restablece el state

print(random.random())    # genera el mismo número
```

```
0.06552885923981311
0.013167991554874137
0.013167991554874137
```

Cada vez que genera un número aleatorio, actualiza el generador, al reestablecerlo el state, vuelve a generar el número anterior. Si generamos otro número aleatorio será diferente

```
print(random.random())
0.83746908209646
```

`random.getrandbits(k)`: Devuelve un entero de Python con k bits aleatorios. Este método se suministra con el generador Mersenne Twister y algunos otros generadores también pueden proporcionarlo como una parte opcional de la API.

```
random.getrandbits(14)
7122

format(random.getrandbits(4), '0b')
'11110111100010'
```

Funciones para enteros

`random.randrange()`: Retorna un elemento de range(start, stop, step) seleccionado aleatoriamente.

```
random.randrange(stop)
random.randrange(start, stop[, step])
```

Ejemplos:

```
random.randrange(10)          # Genera enteros de 0 a 9 inclusive
random.randrange(0, 101, 2)   # Genera enteros de 0 a 100 inclusive
random.randrange(1, 12, 2)    # Las opciones son (1, 3, 5, 7, 9, 11)
random.randrange(0, 11, 2)    # Las opciones son (0, 2, 4, 6, 8, 10)
```

`random.randint(a, b)`: Retorna un entero aleatorio N tal que a <= N <= b. Alias de randrange(a, b+1).

```
random.randint(0, 5)      # Las opciones son (0, 1, 2, 3, 4, 5)
```

Funciones para secuencias

`random.choice(seq)`: Retorna un elemento aleatorio de una secuencia seq no vacía. Si seq está vacía, lanza IndexError.

```
mylist = ["apple", "banana", "cherry"]

print(random.choice(mylist))
banana
```

`random.choices(population, weights=None, *, cum_weights=None, k=1)`: Retorna una lista de elementos de tamaño k elegidos de la population con reemplazo. Si la population está vacía, lanza IndexError. Si se especifica una secuencia weights, las selecciones se realizan de acuerdo con las ponderaciones relativas. Alternativamente, si se da una secuencia cum_weights, las selecciones se harán según los pesos cumulativos.

```
mylist = ["apple", "banana", "cherry"]

print(random.choices(mylist, weights = [10, 10, 10], k = 14))
['banana', 'apple', 'apple', 'cherry', 'cherry', 'cherry',
'apple', 'apple', 'banana', 'banana', 'cherry', 'apple',
'cherry', 'banana']
```

random.shuffle(x[, random]): Baraja la secuencia x en su lugar. El argumento opcional random es una función de 0 argumentos que retorna un flotante random en [0.0, 1.0); por defecto esta es la función random().

```
mylist = ["apple", "banana", "cherry"]
random.shuffle(mylist)

print(mylist)
['apple', 'cherry', 'banana']
```

Para barajar una secuencia inmutable y retornar una nueva lista barajada, utilice sample(x, k=len(x)) en su lugar.

```
cadena = "apple"
random.sample(cadena, k=len(cadena))
['l', 'e', 'p', 'p', 'a']
```

`random.sample(population, k)`: Retorna una lista de longitud k de elementos únicos elegidos de la secuencia de población o conjunto. Se utiliza para el muestreo aleatorio sin reemplazo.

Retorna una nueva lista que contiene elementos de la población sin modificar la población original. La lista resultante está en orden de selección de forma que todos los subsectores

también son muestras aleatorias válidas. Esto permite que los ganadores de la rifa (la muestra) se dividan en primer premio y ganadores del segundo lugar (los subsectores).

```
random.sample([10, 20, 30, 40, 50], k=4)
[20, 30, 40, 10]

mylist = ["apple", "banana", "cherry"]

print(random.sample(mylist, k=2))
['apple', 'banana']
```

Distribuciones para números reales

Las siguientes funciones generan distribuciones específicas para números reales. Los parámetros de la función reciben el nombre de las variables correspondientes en la ecuación de distribución, tal y como se utilizan en la práctica matemática común.; la mayoría de estas ecuaciones se pueden encontrar en cualquier texto estadístico.

Función	Descripción
random.uniform(a, b)	Retorna un número con coma flotante aleatorio N tal que a <= N <= b para a <= b y b <= N <= a para b < a.
random.triangular(low, high, mode)	Retorna un número con coma flotante N tal que low <= N <= high y con el mode especificado entre esos límites. Los límites low (inferior) y high (superior) son por defecto cero y uno. El argumento mode tiene como valor por defecto el punto medio entre los límites, dando lugar a una distribución simétrica.
random.betavariate(alpha, beta)	Distribución beta. Las condiciones de los parámetros son alpha > 0 y beta > 0. Retorna valores dentro del rango entre 0 y 1.
random.expovariate(lambd)	Distribución exponencial. lambd es 1.0 dividido entre la media deseada. Debe ser distinto a cero (El parámetro debería llamarse lambda pero esa es una palabra reservada en Python). Retorna valores dentro del rango de 0 a infinito positivo si lambd es positivo, y de infinito negativo a 0 si lambd es negativo.
random.gammavariate(alpha, beta)	Distribución gamma. (¡No la función gamma!) Las condiciones en los parámetros son alpha > 0 y beta > 0. La función de distribución de la probabilidad es: pdf(x) = (x ** (alpha - 1) * math.exp(-x / beta)) / (math.gamma(alpha) * beta ** alpha)

random.gauss(mu, sigma)	Distribución gaussiana. *mu* es la media y sigma es la desviación estándar. Es un poco más rápida que la función normalvariate().
random.lognormvariate(mu, sigma)	Logaritmo de la distribución normal. Si se usa un logaritmo natural de esta distribución, se obtendrá una distribución normal con media mu y desviación estándar sigma. *mu* puede tener cualquier valor, y sigma debe ser mayor que cero.
random.normalvariate(mu, sigma)	Distribución normal. mu es la media y sigma es la desviación estándar.
random.vonmisesvariate(mu, kappa)	*mu* es el ángulo medio, expresado en radiantes entre 0 y 2*pi, y kappa es el parámetro de concentración, que debe ser mayor o igual a cero. Si kappa es igual a cero, esta distribución se reduce a un ángulo aleatorio uniforme sobre el rango de 0 a 2*pi.
random.paretovariate(alpha)	Distribución de Pareto. alpha es el parámetro de forma.
random.weibullvariate(alpha, beta)	Distribución de Weibull. alpha es el parámetro de escala y beta es el parámetro de forma.

Ejemplos:

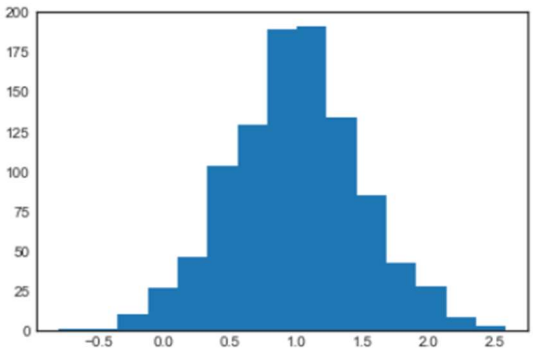
```
random.uniform(2.5, 10.0)   # Random float:  2.5 <= x < 10.0
7.652649592361131

random.uniform(20, 60)
43.54114251593312
```

```
# Ejemplo de una Distribución Gaussiana

import random
import matplotlib.pyplot as plt
%matplotlib inline

campana=[random.gauss(1,0.5) for i in range(1000)]
plt.hist(campana, bins=15)
plt.show()
```



Módulo de matemáticas (**math**)

Python tiene un módulo integrado que puede usar para tareas matemáticas. El módulo de matemáticas tiene un conjunto de métodos y constantes.

`dir(math)`: nos muestra el contenido del módulo, métodos y propiedades disponibles

Métodos

Método	Descripción
math.acos(x)	Devuelve el valor de coseno de arco de x
math.acosh(x)	Devuelve el coseno de arco hiperbólico de x
math.asin(x)	Devuelve el seno de arco de x
math.asinh(x)	Devuelve el seno de arco hiperbólico de x
math.atan(x)	Devuelve el valor de tangente de arco de x
math.atan2(y, x)	Devuelve el arco tangente de y/x en radianes
math.atanh(x)	Devuelve el valor de arco tangente hiperbólico de x
math.ceil(x)	Redondea un número hacia arriba hasta el entero más cercano y devuelve el resultado
math.comb(n, k)	Devuelve el número de formas de elegir k artículos de n elementos sin repetición y orden
math.copysign(x, y)	Devuelve un float que consta del valor del primer parámetro y el signo del segundo parámetro
math.cos(x)	Devuelve el coseno de x
math.cosh(x)	Devuelve el coseno hiperbólico de x
math.degrees(x)	Convierte un ángulo de radianes a grados
math.dist(p, q)	Calcula la distancia euclidiana entre dos puntos especificados (p y q), donde p y q son las coordenadas de ese punto
math.erf(x)	Devuelve la función de error de x
math.erfc(x)	Devuelve la función de error complementario de x
math.exp(x)	Devuelve el valor de Ex, donde E es el número de Euler (aproximadamente 2.718281...), y x es el número que se le pasa
math.expm1(x)	Devuelve el valor de Ex - 1, donde E es el número de Euler (aproximadamente 2.718281...), y x es el número que se le pasa
math.fabs(x)	Devuelve el valor absoluto de un número
math.factorial()	Devuelve el factorial de un número
math.floor(x)	Redondea un número hacia abajo al entero más cercano y devuelve el resultado
math.fmod(x, y)	Devuelve el resto de números especificados cuando un número se divide por otro número
math.frexp()	Devuelve la mantisa y el exponente, de un valor especificado
math.fsum(iterable)	Devuelve la suma de todos los elementos de un iterable (tuplas, matrices, listas, etc.)
math.gamma(x)	Devuelve el valor gamma de x

math.gcd()	Devuelve el valor más alto que puede dividir dos enteros
math.hypot()	Encuentra la distancia euclidiana desde el origen de n entradas
math.isclose()	Comprueba si dos valores están cerca o no
math.isfinite(x)	Comprueba si x es un número finito
math.isinf(x)	Compruebe si x es una infintia positiva o negativa
math.isnan(x)	Comprueba si x es NaN (no un número)
math.isqrt(n)	Devuelve la raíz cuadrada entera más cercana de n
math.ldexp(x, i)	Devuelve la expresión x * 2i donde x es mantisa e i es un exponente
math.lgamma(x)	Devuelve el valor gamma de registro de x
math.log(x, base)	Devuelve el logaritmo natural de un número, o el logaritmo del número a la base
math.log10(x)	Devuelve el logaritmo base-10 de x
math.log1p(x)	Devuelve el logaritmo natural de 1+x
math.log2(x)	Devuelve el logaritmo base-2 de x
math.perm(n, k)	Devuelve el número de formas de elegir k artículos de n artículos con pedido y sin repetición
math.pow(x, y)	Devuelve el valor de x a la potencia de y
math.prod(iterable, *, start=1)	Devuelve el producto de un iterable (listas, matriz, tuplas, etc.)
math.radians(x)	Convierte un valor de grado (x) en radianes
math.remainder(x, y)	Devuelve el valor más cercano que puede hacer que el numerador sea completamente divisible por el denominador
math.sin(x)	Devuelve el seno de x
math.sinh(x)	Devuelve el seno hiperbólico de x
math.sqrt(x)	Devuelve la raíz cuadrada de x
math.tan(x)	Devuelve la tangente de x
math.tanh(x)	Devuelve la tangente hiperbólica de x
math.trunc(x)	Devuelve las partes enteras truncadas de x

Constantes

Constante	Descripción
math.e	Devuelve el número de Euler (2.7182...)
math.inf	Devuelve un infinito positivo de punto flotante
math.nan	Devuelve un valor NaN de punto flotante (not a number)
math.pi	Devuelve PI (3.1415...)
math.tau	Devoluciones tau (6.2831...)

Ejemplos:

```
import math

math.sin(0.5 * math.pi)
```

```
1.0

math.sqrt(x)
7.810249675906654

math.factorial(5)
120
```

*Expresiones Regulares (**RE**)*

Una expresión regular (o RE, por sus siglas en inglés) especifica un conjunto de cadenas que coinciden con ella; las funciones de este módulo permiten comprobar si una determinada cadena coincide con una expresión regular dada (o si una expresión regular dada coincide con una determinada cadena, que se reduce a lo mismo).

El uso de las expresiones regulares en Python viene dado por el paquete re, que hay que importar a nuestro código.

```
import re
```

Algunos de los métodos proporcionados en este paquete son:

Métodos

`re.compile(patrón o pattern, flags=0):` Compila un patrón de una expresión regular en un objeto de expresión regular (RegexObject), que se puede usar para hacer coincidencias usando su match(), search() y otros métodos que repasaremos a continuación. Las operaciones con expresiones regulares que están disponibles como métodos compilados a nivel de módulo, están también como funciones, con algunas diferencias en sus parámetros. Las funciones son atajos que no requieren el paso de la compilación.

```
patron = re.compile('^Python')
patron
re.compile(r'^Python', re.UNICODE)
```

`re.match():` Si hay cero o más caracteres al comienzo de la cadena que coincide con el patrón de expresión regular, devuelva una instancia MatchObject correspondiente. Devuelva None si la cadena no coincide con el patrón.

```
Import re
buscar = re.match("Programa", "Programa en Python")
print(buscar)
<re.Match object; span=(0, 8), match='Programa'>

buscar.string
'Programa en Python'

buscar.group()
'Programa'

buscar.span()
(0, 8)

buscar.start()
0
```

```
buscar.end()
8

enc = re.match("Python", "Programa en Python")
print(enc)
None
```

`re.search(patrón, cadena)`: busca la primera ocurrencia de la expresión regular definida en patrón dentro del string cadena. El resultado se devuelve en un objeto Match en caso de que exista tal ocurrencia, o en un objeto None en caso contrario. >>> import re

```
encontro = re.search("Python", "Programa en Python")
encontro
<re.Match object; span=(12, 18), match='Python'>

encontro.string
"Programa en Python"

encontro.group()
'Python'

encontro.span()
(12, 18)

encontro.start()
12

encontro.end()
18
```


Cuando utilizar search() y match()

1. Si desea ubicar **una coincidencia en cualquier lugar de la cadena**, **use search()**.
2. Si necesita **hacer coincidir al principio de la cadena**, o para **hacer coincidir toda la cadena**, **use match**. Es más rápido.

`re.findall(patrón, cadena)`: devuelve una lista que contiene todas las ocurrencias de la expresión regular definida en patrón dentro del string cadena. Las ocurrencias se devuelven en el mismo orden en que se han encontrado.

```
import re
re.findall("Python", "En este curso de Python, aprenderemos a
programar en Python y nos convertiremos en Pythonista.")
['Python', 'Python', 'Python']
```

`finditer()`: El cual es similar a `re.findall`, pero en lugar de devolvernos una lista nos devuelve un iterador.

```
encontrados = re.finditer("Python", "En este curso de Python,
aprenderemos a programar en Python y nos convertiremos en
Pythonista.")

print(encontrados)

for encontro in encontrados:
    print(encontro.group(0), encontro.span() )

<callable_iterator object at 0x000001625BFFDCC8>
Python (17, 23)
Python (53, 59)
Python (83, 89)
```

`re.split(separador, cadena)`: divide la cadena tomando en cuenta las ocurrencias del separador. El resultado se devuelve en una lista.

```
correo = re.split("@","m.sierra@cloudftic.com")
correo
['m.sierra', 'cloudftic.com']

correo[0]
'm.sierra'

correo[1]
'cloudftic.com'
```

`re.sub(patrón, repl, string)`: reemplaza las coincidencias con el texto de su elección (repl).

```
import re

txt = "Se esperan abundantes lluvias"
x = re.sub("\s", "-", txt)
print(x)
Se-esperan-abundantes-lluvias
```

Al buscar direcciones de correo electrónico, números de teléfono, validar campos de entrada, o una letra mayúscula seguida de dos minúsculas y de 5 dígitos entre 1 y 3; podemos utilizar los meta caracteres para crear nuevos patrones.

Metacaracteres

Se conoce como meta caracteres aquellos que, dependiendo del contexto, tienen un significado especial para las expresiones regulares. Por lo tanto, debemos usar el carácter de barra inversa ('\') para indicar formas especiales o para permitir el uso de caracteres especiales sin invocar su significado especial.

REGEX Sintaxis	Descripción	Ejemplo	Coincidencia	No Coincidencia
^	Comienza con	^demo	demostración	mi demostración
$	Finaliza con	demo$	mi demostración	demostración
.	Cualquier carácter	go.gle	google	gogle
*****	Indica la repetición de un carácter cero o más veces	goo*gle	gogle, goooogle	goggle
+	Indica la repetición de un carácter una o más veces	goo+gle	google, goooogle	gogle, goggle
?	Indica la repetición de un carácter cero o una vez	demos?123	demo123, demos123	demoA123
[abc]	Cualquiera de estos caracteres	analy[zs]e	analyse, analyze	analyxe
[a-c]	Cualquier carácter del rango	demo[2-4]	demo2, demo3	demo1, demo5
[^abc]	Ninguno de estos caracteres	analy[^zs]e	analyxe	analyse, analyze
[^a-c]	Ningún carácter del rango	demo[^2-4]	demo1, demo5	demo2, demo3
|	o	demo|example	demo, demos, example	test
{n}	n veces exactas	w[3]	www	w.ww
{n,m}	Desde n hasta m veces	a[4,7]	aaaa, aaaaa, aaaaaa, aaaaaaa	aaaaaaaa, aaa, a
{n,}	Al menos n veces	go[2,]gle	google, gooogle, goooogle	ggle, gogle
[]	Grupo	^[demo|example][0-9]+	demo1, example4	demoexample2
****	Escape	AU\$10	AU$10, AU$100	AU10, 10
\s	Espacios en blanco			
\S	No espacios en blanco			
\d	Dígitos			
\D	No dígitos			
\w	Caracteres			
\W	No caracteres [espacios, signos de puntuación]			

Inicio y fin de la extracción: ()

Los paréntesis ( ) no forman parte del patrón a comprobar, pero indican respectivamente dónde empieza y termina la extracción del texto. Un caso de uso es la extracción de dominios en direcciones de correo electrónico. Esta operación la podemos realizar mediante el patrón:

‘@([^]*)’ En este caso sabemos que el dominio viene después de un símbolo de arroba (@) que indicamos en nuestra expresión regular, seguido de una condición cerrada entre paréntesis ya que no queremos que el resultado contenga arrobas.

```
frase = "Tengo dos correos electrónicos que son
          m.sierra@cloudftic.com y maria.c.sierra.f@gmail.com"
patron = '@([^ ]*)'
re.findall(patron, frase)
['cloudftic.com', 'gmail.com']
```

Si cambiamos la posición del símbolo de arroba (@)

```
patron = '([^ ]*)@'
re.findall(patron, frase)
['m.sierra', 'maria.c.sierra.f']
```

Ejemplos:

```
frase = "Python es un lenguaje de programación."
patron = '^Python'
re.findall(patron, frase)
['Python']


frase = "Me gusta aprender Python y programar en Python"
patron = 'Python$'
re.findall(patron, frase)
['Python']


frase = "Cython no es ningún lenguaje de programación y Nython
tampoco pero Python sí"

patron = '^.ython'
re.findall(patron, frase)
['Cython']


patron = '\\s.ython'
re.findall(patron, frase)
[' Nython', ' Python']


patron = '[CN]ython'
re.findall(patron, frase)
['Cython', 'Nython']


patron = '[A-Z][a-z]+'
```

```
re.findall(patron, frase)
['Cython', 'Nython', 'Python']


frase = "Tengo 2 hijos que tienen 18 y 15 años"
patron = '[0-9]+'
re.findall(patron, frase)
['2', '18', '15']


frase = "¡Esto es una frase! Además contiene signos de puntuación.
¿Los eliminamos?"
patron = '[^¡!.¿? ]+'
re.findall(patron, frase)
['Esto', ‘es’,  'una',  'frase', 'Además', 'contiene', 'signos',
'de', 'puntuación', 'Los', 'eliminamos']
```