Tema 1:
Python 5

Índice

1. Módulos y paquetes

Es posible hacer librerías de funciones y clases metiéndolas en archivos (módulos) y estos archivos en directorios y subdirectorios (paquetes).

1.1. Módulos

Un módulo en Python es un fichero .py que alberga un conjunto de funciones, variables o clases y que puede ser usado por otros módulos.

Los módulos se importan con la palabra reservada import (se importa el nombre del fichero sin la extensión .py)

```
import nombreFichero
```

Esto hace que tengamos acceso todas las funciones de ese módulo. Por ejemplo:

```
# mimodulo.py
def suma(a, b):
    return a + b

def resta(a, b):
    return a - b
```

Y en otro fichero:

```
# otromodulo.py
import mimodulo

print(mimodulo.suma(4, 3))   # 7
```

```
7
```

```
print(mimodulo.resta(10, 9)) # 1
```

```
1
```

Si solo queremos importar alguna de las funciones de un módulo podemos usar:

```
from mimodulo import suma
```

En este caso solo podríamos usar la función suma del módulo mimodulo.

De este modo, no sería necesario usar el nombre del módulo para acceder a la parte que hemos importado.

Con la misma sintaxis podemos importar todo el módulo y no tener que usar el nombre del módulo para acceder a sus partes.

```
form mimodulo import * # importa todo
```

Aunque el hecho de poner el nombre del módulo delante de la función es bueno por claridad y para evitar sobreescribir funciones con el mismo nombre.

También Podemos cambiar el nombre de un módulo o de una parte de un módulo con la palabra reservada as.

No es una práctica recomendada, puesto que dificulta la lectura del código, pero puede ser útil para evitar solapamientos de nombres.

```
import math as m
print(m.pi)
```

```
3.141592653589793
```

1.2. __name__

__name__ es una variable interna de Python a la que se le asigna el nombre del módulo que se ejecuta:

- Si es el módulo principal: se le asgina el nombre __main__.
- Si es un módulo importando con import se asigna el nombre del modulo.

Por ejemplo

```
# mimodulo.py
def suma(a, b):
    print (f"Dentro: {__name__}")
    return a + b
```

```
import mimodulo
print (f"Fuera: {__name__}")
print(mimodulo.suma(4, 3))   # 7
```

```
Fuera: __main__
Dentro: mimodulo
7
```

1.3. Paquetes

Un paquete es un directorio que contiene archivos de módulos de Python.

Además para que dicho directorio pueda ser paquete debe contener en el archivo __init__.py que puede estar vacio o contener otra configuraciones en las que no entraremos por el momento.

Para importar un paquete se usa la sintaxis.

```
import paquete
```

Para importar un módulo de un paquete usamos la sintaxis:

```
from paquete import modulo
```

o

```
import paquete.modulo.
```

Los módulos se buscarán en el directorio del módulo principal y en los directorios que se encuentren en el path sys.path.

Para ver que directorios incluye sys.path se puede usar

```
import sys
print(sys.path)
```

Y añadir un directorio al sys.path con el siguiente código:

```
import sys
sys.path.append("nuevo_directorio")
```

2. Ficheros

Antes de trabajar con un fichero tenemos que abrirlo. Para ello tenemos la función open que recibe como parámetro el nombre del archivo a abrir y el modo en que se va a abrir.

El modo puede ser:

Modo	Tipo	Descripción
r	Read	Abre el archivo en modo lectura. El archivo tiene que existir previamente, en caso contrario se lanzará una excepción de tipo IOError
w	Write	Abre el archivo en modo escritura. Si el archivo no existe se crea. Si existe, sobreescribe el contenido.
a	Append	Abre el archivo en modo escritura. Se diferencia del modo 'w' en que en este caso no se sobreescribe el contenido del archivo, sino que se comienza a escribir al final del archivo.
r+	Read-Write	permite lectura y escritura simultáneas.

Si no se especifica el modo, por defecto se abre en modo lectura.

```
fichero = open("fichero.txt", "r", encoding="utf-8")
contenido = fichero.read()
print(contenido)
```

Al terminar de usar un fichero este se ha de cerrar para liberar recursos. Para ello se utilizar la función close que recibe como parámetro el nombre del archivo a cerrar.

```
fichero.close()
```

También es posible utilizar un archivo dentro de un bloque with. Esto permite que al finalizar el bloque el archivo se cierra automáticamente sin usar close.

```
with open("fichero.txt", "r", encoding="utf-8") as fichero:
    print(fichero.read())
```

Métodos de lectura

Para leer datos de un fichero tenemos los métodos siguientes:

- fichero.read(): lee todo el archivo y lo devuelve como un string.
- fichero.read(n): lee n bytes.
- fichero.readline(): lee una línea del archivo.
- fichero.readlines(): lee todas las líneas del archivo y las devuelve como una lista.
- for linea in fichero: lee el archivo línea por línea.
- fichero.readlines(n): lee n líneas del archivo y las devuelve como una lista.
- fichero.readline(n): lee n caracteres del archivo y los devuelve como un string.

```
fichero = open("fichero.txt", "r", encoding="utf-8")
with open("fichero.txt", "r", encoding="utf-8") as fichero:
    for linea in fichero:
        print(linea)
```

Métodos de escritura

Para escribir datos en un fichero tenemos los siguientes métodos:

- fichero.write(cadena): escribe la cadena que se pasa como parámetro.
- fichero.writelines(lista): escribe la lista de cadenas que se pasa como parámetro.

Acceso aleatorio

- seek(posicion,[inicio]): Sitúa el puntero del archivo donde indique posición. Si establecemos el parámetro inicio a 0, la posición se refiere desde el principio (funcionamiento por defecto), si colocamos 1 se refiere a posición desde la actual y 2 desde el final.
- tell(): Me dice la posición del puntero. Distancia en bytes desde el principio.

2.1. Tratamiento de excepciones

Para tratar las excepciones que se pueden producir al abrir o cerrar un archivo, se utiliza un bloque try-except.

- Si abrimos un archivo que no existe, se producirá una excepción de tipo FileNotFoundError .
- Si intentamos leer, escribir o cerrar un archivo que ya está cerrado, se producirá una excepción de tipo ValueError .
- Si intentamos abrir un archivo que ya está abierto, se producirá una excepción de tipo OSError .
- Si intentamos escribir en un archivo que está abierto en modo lectura, o leer un archivo que se ha abierto en modo escritura, se producirá una excepción de tipo UnsupportedOperation .

```
try:
    fichero = open("fichero.txt", "r")
    # Operaciones con el fichero
except FileNotFoundError:
    print("Error al abrir el archivo")
finally:
    fichero.close()
```

3. Fechas y hora

Usaremos el módulo datetime.

```
from datetime import datetime, date, timedelta

# Día actual
hoy = date.today()
print(hoy)

# Fecha y hora actual
ahora = datetime.now()
print(ahora)
```

Tenemos disponibles las siguientes tipos:

Tipo	Descripción
date	Fecha representada por año, mes y día
time	Hora representada por horas, minutos, segundos y microsegundos
datetime	Combinación de una fecha y una hora. Representados por un año, mes, día, horas, minutos, segundos y microsegundos
timedelta	Diferencia entre dos date o datetime

Atributos de date y datetime.

Atributo	Descripción
year	Año
month	Mes
day	Día
hour	Hora
minute	Minuto
second	Segundo
microsecond	Microsegundo
tzinfo	Información de la zona horaria
weekday	Día de la semana (lunes es 0)

Para crear una nueva fecha podemos usar los siguientes constructores:

- datetime: al que se le pasa el año, mes, día, hora, minutos, segundos y milisegundos.

```
fecha_hora = datetime(2024, 10, 15, 12, 30, 45)
```

- date: al que se le pasa el año, mes, día.

```
fecha = datetime(2024, 10, 15)
```

Si los valores no son correctos se produce una excepción.

Calcular fechas

Las fechas se pueden restar:

```
fecha=datetime(2023,12,10,10,20,30)
fecha2=datetime(2023,12,12)

fecha=fecha-fecha2
print(fecha)
```

```
-2 days, 10:20:30
```

Si lo que queremos a una fecha sumar o restar un intervalo se usa timedelta.

Puede usar las siguientes unidades que pueden ser enteros o floats y positivos o negativos.

- days
- seconds
- microseconds
- milliseconds
- minutes
- hours
- weeks

```
fecha = date.today()
pasado_mañana = fecha + timedelta(days=2)
```

Formatear fechas

Para formatear fechas, utilizamos strftime(), que recibe un formato de fecha.

```
now = datetime.now()
format = now.strftime('Día :%d, Mes: %m, Año: %Y, Hora: %H, Minutos: %M, Segundos: %S')
```

Para ver todas los directivas disponibles se puede consultar: https://strftime.org/

Otros métodos relacionados

El módulo calendar nos permite trabajar con calendarios.

```
import calendar

calendario = calendar.month(2024, 1)
print(calendario)

print(calendar.isleap(2024))

print(calendar.calendar(2024))
```

4. Otros módulos.

Vamos a ver otros módulos que incorpora Python.

Para ver todos los métodos disponibles se puede usar la función dir.

```
import math
print(dir(math))        #muestra el listado de funciones del módulo math
help (math.sqrt)        #muestra la ayuda de la función sqrt del módulo math
```

- math: ofrece funciones matemáticas.

```
import math
sqrt_val = math.sqrt(64)
pi_const = math.pi
print(sqrt_val)
print(pi_const)
```

- random: utilizado para generar números aleatorios

```
import random
num = random.randint(1, 10)
print(f"Random integer between 1 and 10: {num}")
fruits = ["Java", "C", "C++", "Python"]
chosen_fruit = random.choice(fruits)
```

- os: Se utiliza para interactuar con el sistema operativo y ofrece funcionalidades a nivel de sistema operativo. Por ejemplo, interactuar con el sistema de archivos, leer directorios y lanzar aplicaciones.

```
import os
directory = os.getcwd()
print(directory)
os.system('clear' if os.name == 'posix'  else 'cls') # Se borra la pantalla
```

- os.path: permite trabajar que rutas y con archivos u directorios.

```
import os
import os.path as p
for f in os.listdir("F:"):
    print(f, p.isfile(f))
```

- Sys: proporciona funciones y variables que interactúan con el entorno de ejecución de Python. Permite a los desarrolladores acceder a los atributos del intérprete de Python y manipularlos. Ofrece una serie de otras funcionalidades, como el acceso a flujos de entrada/salida, el acceso a información de memoria, etc.

```
import sys
print("Python version:", sys.version)
print("Command line arguments:", sys.argv)
sys.exit(1)
```

- Tinker: es la biblioteca GUI (Graphical User Interface) estándar de Python.

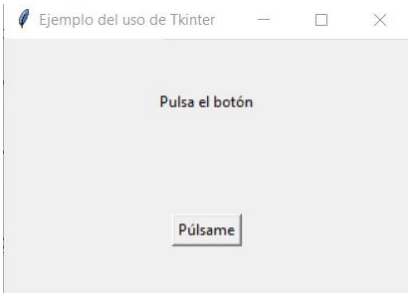
```
import tkinter as tk
def on_button_click():
    label.config(text="Hola!!!!")

root = tk.Tk()
root.title("Ejemplo del uso de Tkinter")
root.geometry("600x480")

label = tk.Label(root, text="Pulsa el botón")
label.pack(pady=40)

button = tk.Button(root, text="Púlsame", command=on_button_click)
button.pack(pady=40)

root.mainloop()
```



Para ver una lista completa de módulos se pueden consultar el siguiente enlace: https://docs.python.org/3/py-modindex.html

5. Ejercicios

1. Dado un fichero y un carácter cuente el número de ocurrencias de ese carácter en el fichero.
2. Dado un fichero encuentre el carácter más usado.
3. Dado un fichero muestre las líneas donde aparece una cadena de texto, pasada como parámetro, junto con el número de línea en la que aparece.
4. Crea un programa que realice las siguientes acciones:
   - Dividir un fichero en función de:
     - De un número n de caracteres (en cada fichero generado debe poseer n caracteres).
     - De un número l de líneas (en cada fichero generado debe poseer l líneas).
   - Unir ficheros: dada una lista de ficheros generará un nuevo fichero que resultara de la unión de los anteriores.
5. Crea un programa que, para un directorio dado, muestre, de forma separada, los ficheros y directorios que posee.
6. Crea una función que pasándole un año, mes y día escriba en un fichero todas las fechas, de lunes a domingo, de la semana en la que caiga esa fecha.
7. Crea una función que pasándole una fecha de nacimiento nos diga, en modo texto, en que día nació.

6. Ejercicios extra

8. Implementa juego del wordle: https://lapalabradeldia.com/

   Para los colores puedes usar los siguientes códigos ANSI.

```
BLACK = "\033[0;30m"
RED = "\033[0;31m"
GREEN = "\033[0;32m"
BROWN = "\033[0;33m"
BLUE = "\033[0;34m"
PURPLE = "\033[0;35m"
CYAN = "\033[0;36m"
LIGHT_GRAY = "\033[0;37m"
DARK_GRAY = "\033[1;30m"
LIGHT_RED = "\033[1;31m"
LIGHT_GREEN = "\033[1;32m"
YELLOW = "\033[1;33m"
LIGHT_BLUE = "\033[1;34m"
LIGHT_PURPLE = "\033[1;35m"
LIGHT_CYAN = "\033[1;36m"
LIGHT_WHITE = "\033[1;37m"
BOLD = "\033[1m"
FAINT = "\033[2m"
ITALIC = "\033[3m"
UNDERLINE = "\033[4m"
BLINK = "\033[5m"
NEGATIVE = "\033[7m"
CROSSED = "\033[9m"
END = "\033[0m"
```

   Por ejemplo

```
print(f"C{GREEN}A{RED}S{LIGHT_WHITE}A")
   CASA
```

9. Programar el juego mini hundir la flota contra el ordenador.

   - Se debe generar una coordenada (de un tablero de 10x10 posiciones) aleatoria (sin mostrar nada en pantalla) y guardarla (barco del ordenador).
   - Luego el usuario debe introducir una coordenada que el ordenador intentará averiguar (barco del usuario).
   - A partir de ese momento y por turnos el usuario meterá coordenadas con la idea de localizar el barco del ordenador.
   - Tras su turno será el ordenador el que diga una coordenada aleatoria y comprobará si coincide con la del barco del usuario. No es necesario comprobar si el ordenador repite coordenadas. Lo que sí es necesario es que el usuario pueda introducir coordenadas del estilo de A, 4 con letras mayúsculas o minúsculas (pide primero la letra y luego el número de la coordenada).
   - Se debe hacer programación modular.