Módulos y Scripts en Python

¿Qué es un módulo?

Un **módulo** en Python es simplemente un archivo `.py` que contiene código Python (funciones, clases, variables). Los módulos permiten organizar y reutilizar código.

Crear un módulo

Crea un archivo, por ejemplo `matematicas.py`:

```python
# matematicas.py
def sumar(a, b):
    return a + b

def restar(a, b):
    return a - b

PI = 3.14159
```

Importar módulos

Importar todo el módulo

```python
import matematicas

resultado = matematicas.sumar(5, 3)
print(matematicas.PI)
```

Importar funciones específicas

```python
from matematicas import sumar, PI

resultado = sumar(5, 3)
print(PI)
```

Importar con alias

```python
import matematicas as mat

resultado = mat.sumar(5, 3)
```

Importar todo (no recomendado)

```python
from matematicas import *
```

El bloque `if __name__ == "__main__"`

Este es un patrón fundamental para diferenciar cuando un archivo se ejecuta como script vs. cuando se importa como módulo.

¿Qué es `__name__`?

- Cuando ejecutas un archivo directamente: `__name__ == "__main__"`
- Cuando importas un archivo como módulo: `__name__ == "nombre_del_modulo"`

Ejemplo práctico

```python
# mi_script.py

def saludar(nombre):
    return f"Hola, {nombre}!"

def main():
    print(saludar("Mundo"))
    print("Este código solo se ejecuta como script")

# Este bloque solo se ejecuta si el archivo se ejecuta directamente
if __name__ == "__main__":
    main()
```

Uso: - Como script: `python mi_script.py` → ejecuta `main()` - Como módulo: `import mi_script` → solo define las funciones, no ejecuta `main()`

Ejecutar programas como scripts

Método 1: Directamente con Python

```
python mi_script.py
python3 mi_script.py  # En sistemas con ambas versiones
```

Método 2: Con argumentos

```python
# script_con_args.py
import sys

if __name__ == "__main__":
    print(f"Argumentos recibidos: {sys.argv}")
    if len(sys.argv) > 1:
        print(f"Primer argumento: {sys.argv[1]}")
```

```
python script_con_args.py argumento1 argumento2
```

Método 3: Scripts ejecutables (Linux/Mac)

Añade el shebang al inicio del archivo:

```
#!/usr/bin/env python3

def main():
    print("Script ejecutable")

if __name__ == "__main__":
    main()
```

Hazlo ejecutable:

```
chmod +x mi_script.py
./mi_script.py
```

Módulo `-m`

Ejecutar módulos como scripts usando el flag `-m`:

```
python -m http.server 8000  # Servidor HTTP
python -m json.tool archivo.json  # Formatear JSON
python -m pip install paquete  # Instalar paquetes
```

Estructura típica de un script

```
#!/usr/bin/env python3
"""
Descripción del script
"""

import modulo1
import modulo2

# Constantes globales
CONSTANTE = 100

# Funciones
def funcion1():
    pass

def funcion2():
    pass

# Función principal
def main():
    """Punto de entrada del programa"""
```

```
    print("Ejecutando programa...")
    funcion1()
    funcion2()

# Punto de entrada
if __name__ == "__main__":
    main()
```

Rutas de búsqueda de módulos

Python busca módulos en:

1. El directorio del script actual
2. Directorios en la variable de entorno `PYTHONPATH`
3. Directorios de instalación estándar

Verificar rutas:

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

Añadir rutas personalizadas:

```
import sys
sys.path.append('/ruta/a/mis/modulos')
```

Paquetes

Un **paquete** es un directorio que contiene módulos y un archivo `__init__.py`:

```
mi_paquete/
    __init__.py
    modulo1.py
    modulo2.py
    subpaquete/
        __init__.py
        modulo3.py
```

Importar desde paquetes:

```
from mi_paquete import modulo1
from mi_paquete.subpaquete import modulo3
```

Copias de seguridad en Python

1. Copiar archivos individuales con `shutil`

```
import shutil
import os
```

```python
from datetime import datetime

def hacer_backup_archivo(origen, carpeta_backup='backups'):
    """Crea una copia de seguridad de un archivo"""
    # Crear carpeta de backups si no existe
    if not os.path.exists(carpeta_backup):
        os.makedirs(carpeta_backup)

    # Generar nombre con timestamp
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    nombre_archivo = os.path.basename(origen)
    nombre_backup = f"{nombre_archivo}.{timestamp}.bak"

    destino = os.path.join(carpeta_backup, nombre_backup)

    # Copiar archivo
    shutil.copy2(origen, destino)
    print(f"Backup creado: {destino}")
    return destino
```

Uso:

```python
hacer_backup_archivo('datos.txt')
hacer_backup_archivo('config.json', 'mis_backups')
```

2. Copiar directorios completos

```python
def hacer_backup_directorio(origen, destino=None):
    """Crea una copia de seguridad de un directorio completo"""
    if destino is None:
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        destino = f"{origen}_backup_{timestamp}"

    # Copiar todo el árbol de directorios
    shutil.copytree(origen, destino)
    print(f"Backup del directorio creado: {destino}")
    return destino
```

Uso:

```python
hacer_backup_directorio('mi_proyecto')
hacer_backup_directorio('mi_proyecto', 'backups/proyecto_20241023')
```

3. Crear archivos ZIP de backup

```python
import zipfile

def crear_backup_zip(origen, nombre_zip=None):
```

```
    """Crea un archivo ZIP de backup"""
    if nombre_zip is None:
        timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
        nombre_zip = f"backup_{timestamp}.zip"

    with zipfile.ZipFile(nombre_zip, 'w', zipfile.ZIP_DEFLATED) as zipf:
        if os.path.isfile(origen):
            # Backup de un solo archivo
            zipf.write(origen, os.path.basename(origen))
        else:
            # Backup de directorio
            for root, dirs, files in os.walk(origen):
                for file in files:
                    ruta_completa = os.path.join(root, file)
                    ruta_relativa = os.path.relpath(ruta_completa, origen)
                    zipf.write(ruta_completa, ruta_relativa)

    print(f"Backup ZIP creado: {nombre_zip}")
    return nombre_zip
```

Uso:

```
crear_backup_zip('datos.txt')
crear_backup_zip('mi_proyecto', 'backups/proyecto.zip')
```

4. Sistema de backups rotativos

```
def backup_rotativo(origen, carpeta_backup='backups', max_backups=5):
    """Mantiene solo las últimas N copias de seguridad"""
    if not os.path.exists(carpeta_backup):
        os.makedirs(carpeta_backup)

    # Crear nuevo backup
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    nombre_base = os.path.basename(origen)
    nombre_backup = f"{nombre_base}.{timestamp}.bak"
    destino = os.path.join(carpeta_backup, nombre_backup)

    shutil.copy2(origen, destino)

    # Obtener todos los backups del mismo archivo
    patron = f"{nombre_base}.*.bak"
    backups = []
    for archivo in os.listdir(carpeta_backup):
        if archivo.startswith(nombre_base) and archivo.endswith('.bak'):
            ruta = os.path.join(carpeta_backup, archivo)
            backups.append((os.path.getmtime(ruta), ruta))
```

```
    # Ordenar por fecha (más antiguos primero)
    backups.sort()

    # Eliminar backups antiguos si exceden el máximo
    while len(backups) > max_backups:
        _, ruta_antigua = backups.pop(0)
        os.remove(ruta_antigua)
        print(f"Backup antiguo eliminado: {ruta_antigua}")

    print(f"Backup creado: {destino}")
    return destino
```

Uso:

```
backup_rotativo('datos.txt', max_backups=3)
```

Cargar/Restaurar copias de seguridad

1. Restaurar archivo desde backup

```
def restaurar_backup(archivo_backup, destino=None):
    """Restaura un archivo desde una copia de seguridad"""
    if destino is None:
        # Extraer nombre original del backup
        nombre = os.path.basename(archivo_backup)
        # Remover timestamp y .bak
        partes = nombre.rsplit('.', 2)
        destino = partes[0] if len(partes) > 1 else nombre

    shutil.copy2(archivo_backup, destino)
    print(f"Archivo restaurado: {destino}")
    return destino
```

Uso:

```
restaurar_backup('backups/datos.txt.20241023_153000.bak')
restaurar_backup('backups/datos.txt.20241023_153000.bak', 'datos_recuperados.txt')
```

2. Listar backups disponibles

```
def listar_backups(carpeta_backup='backups', archivo=None):
    """Lista todos los backups disponibles"""
    if not os.path.exists(carpeta_backup):
        print("No hay carpeta de backups")
        return []
```

```
    backups = []
    for nombre in os.listdir(carpeta_backup):
        ruta = os.path.join(carpeta_backup, nombre)
        if os.path.isfile(ruta):
            if archivo is None or nombre.startswith(archivo):
                fecha = datetime.fromtimestamp(os.path.getmtime(ruta))
                tamaño = os.path.getsize(ruta)
                backups.append({
                    'nombre': nombre,
                    'ruta': ruta,
                    'fecha': fecha,
                    'tamaño': tamaño
                })

    # Ordenar por fecha (más recientes primero)
    backups.sort(key=lambda x: x['fecha'], reverse=True)

    print(f"\n{'Archivo':<40} {'Fecha':<20} {'Tamaño (bytes)'}")
    print("-" * 70)
    for b in backups:
        print(f"{b['nombre']:<40} {b['fecha']:%Y-%m-%d %H:%M:%S} {b['tamaño']:>10}")

    return backups
```

Uso:

```
listar_backups()  # Todos los backups
listar_backups(archivo='datos.txt')  # Solo backups de datos.txt
```

3. Restaurar desde ZIP

```
def restaurar_desde_zip(archivo_zip, destino='.'):
    """Extrae archivos de un backup ZIP"""
    with zipfile.ZipFile(archivo_zip, 'r') as zipf:
        zipf.extractall(destino)
        print(f"Archivos restaurados en: {destino}")
        print("Archivos extraídos:")
        for nombre in zipf.namelist():
            print(f"  - {nombre}")
```

Uso:

```
restaurar_desde_zip('backups/proyecto.zip')
restaurar_desde_zip('backups/proyecto.zip', 'proyecto_restaurado')
```

4. Script completo de backup automático

```python
#!/usr/bin/env python3
"""
Script de backup automático
Uso: python backup_manager.py [crear|listar|restaurar]
"""

import sys

def main():
    if len(sys.argv) < 2:
        print("Uso: python backup_manager.py [crear|listar|restaurar]")
        return

    comando = sys.argv[1]

    if comando == 'crear':
        if len(sys.argv) < 3:
            print("Uso: python backup_manager.py crear <archivo>")
            return
        archivo = sys.argv[2]
        backup_rotativo(archivo)

    elif comando == 'listar':
        archivo = sys.argv[2] if len(sys.argv) > 2 else None
        listar_backups(archivo=archivo)

    elif comando == 'restaurar':
        if len(sys.argv) < 3:
            print("Uso: python backup_manager.py restaurar <backup>")
            return
        backup = sys.argv[2]
        destino = sys.argv[3] if len(sys.argv) > 3 else None
        restaurar_backup(backup, destino)

    else:
        print(f"Comando desconocido: {comando}")

if __name__ == "__main__":
    main()
```

Uso:

```
python backup_manager.py crear datos.txt
python backup_manager.py listar
python backup_manager.py restaurar backups/datos.txt.20241023_153000.bak
```

Buenas prácticas

Para módulos y scripts:

- Usa `if __name__ == "__main__":` en scripts reutilizables
- Documenta tus módulos con docstrings
- Usa nombres descriptivos para módulos (minúsculas, guiones bajos)
- Evita imports circulares
- Agrupa código relacionado en el mismo módulo
- Evita `from modulo import *` (contamina el namespace)

Para backups:

- Usa timestamps en los nombres de backup
- Implementa backups rotativos para ahorrar espacio
- Comprime backups grandes con ZIP
- Guarda backups en ubicaciones diferentes (disco externo, nube)
- Prueba regularmente que los backups se pueden restaurar
- Documenta qué archivos/directorios son críticos
- No sobreescribas backups sin confirmación
- No guardes backups solo en el mismo disco

Recursos adicionales

- Documentación oficial de Python - Módulos
- shutil - Operaciones de archivos de alto nivel
- zipfile - Trabajar con archivos ZIP
- os.path - Manipulación de rutas