5. Excepciones, módulos y entrada/salida

Índice

1. Manejo de excepciones

Una excepción es un error en tiempo de ejecución, que normalmente no es previsible en tiempo de codificación, de ahí que haya que "manejar" la excepción en caso de que ocurra:

```
try:
  print(x)
except:
  print("Ocurrió una excepción genérica")
```

Se pueden manejar los distintos tipos de excepciones predefinidas de forma separada:

```
try:
  print(x)
except NameError:
  print("No existe esa variable")
except:
  print("Ocurrió una excepción genérica")
```

Realizar alguna operación en caso de que no haya excepciones que manejar:

```
try:
  print(x)
except:
  print("Ocurrió una excepción genérica")
else:
  print("Todo va bien")
```

O realizar alguna operación en cualquiera de los casos (con o sin excepciones):

```
try:
  print(x)
except:
  print("Ocurrió una excepción genérica")
```

```
finally:
  print("Todo terminó")
```

Al igual que en los condicionales y bucles, la anidación está permitida para distintas operaciones dependientes entre sí:

```
try:
  f = open("dummy.txt")
  try:
    f.write("En un lugar de La Mancha")
  except:
    print("El fichero no se pudo escribir")
  finally:
    f.close()
except:
  print("El fichero no se pudo abrir")
```

Por último, en tiempo de codificación se puede establecer el lanzamiento de excepciones voluntariamente:

```
x = "hello"

if not type(x) is int:
  raise TypeError("No es un número entero")
```

2. Gestión de paquetes

Creación de módulos propios

Un módulo es un fichero .py que agrupa un conjunto de funciones:

```
def add(number1, number2):
  return number1 + number2
def substract(number1, number2):
  return number2 - number1
```

Que después se pueden reutilizar:

```
import mycalculator

result = mycalculator.add(1,2)
```

O con un alias:

```
import mycalculator as calc

result = calc.add(1,2)
```

Permite acceso exterior a sus variables:

```
# ---- myHeroes.py ----

myHero1 = {
  "name": "V",
  "age": 42,
  "country": "UK"
}

# ---- otherFile.py ----

import myHeroes

heroCountry = myHeroes.myHero1["country"]
print(heroCountry)
```

O preseleccionarlas en la importación:

```
from myHeroes import myHero1

heroCountry = myHero1["country"]
print(heroCountry)
```

Módulos estándar de Python

El lenguaje aprovisiona una extensa colección de módulos de serie con diversas funcionalidades del mismo tipo para cada módulo:

```
import platform

x = platform.system()
print(x)
```

Existen multitud de módulos estándar, algunos de los más usados como ejemplos podrían ser: datetime, math, re, sqlite3, etc.

Importación de paquetes

Los módulos se despliegan en forma de paquetes para que puedan compartirse y ser usados según las necesidades de cada proyecto.

Para poder usarlos, es necesario un gestor de paquetes que se encarga de su instalación y desinstalación, en este caso pip:

```
pip install camelcase

pip uninstall camelcase
```

Y después se usa como un módulo más:

```
import camelcase

c = camelcase.CamelCase()
txt = "hello world"

print(c.hump(txt))
```

Se pueden también listar paquetes instalados:

```
pip list
```

Crear un fichero con la lista de paquetes instalados y sus versiones actuales:

```
pip freeze > requirements.txt
```

Que se puede reutilizar para instalar rápidamente esos mismos paquetes en otro entorno:

```
pip install -r requirements.txt
```

3. Entrada y salida de datos

Ficheros

Las operaciones de lectura y escritura en disco se realizan en ficheros, que son estructuras de datos secuenciales de almacenamiento persistente.

Para empezar a trabajar con un fichero, es necesario abrirlo especificando el modo de acceso:

- Creación de un fichero vacío (create): "x"
- Lectura de un fichero existente (read): "r"

- Escritura desde el principio (write): "w"
- Escritura desde el final (append): "a"

Y el tipo de fichero:

- Texto: "t" (valor por defecto)
- Binario: "b"

Mediante el método open(), indicando la ruta (relativa o absoluta):

```
myFile = open("dummy.txt", "r")
```

Es posible hacer lectura completa, de una cantidad de bytes (caracteres si es de texto) o líneas:

```
allFile = myFile.read()
beginning = myFile.read(8)
firstLine = myFile.readline()
```

Finalmente y tras realizar las operaciones deseadas, el fichero debe cerrarse:

```
myFile.close()
```

Por línea de comandos

Otra forma de proporcionar una entrada de datos a los scripts de Python es mediante la consola de comandos, al llamar al intérprete para ejecutar el código de un script:

```
python myscript.py hola mundo
```

Cuyo contenido sea, por ejemplo, el siguiente:

```
import sys

print(f"Mi script se llama: {sys.argv[0]}")
print(f"Y tiene {len(sys.argv)} parámetros")
print(f"Que son estos: {sys.argv}")
```

4. Serialización de objetos

Serializar una lista en un fichero:

```
import pickle

myFile = open("animalsFile.dat", "wb")

animals = ["python", "monkey", "camel"]

pickle.dump(animals, myFile)

myFile.close()
```

Deserializar una lista de un fichero:

```
import pickle

myFile = open("animalsFile.dat", "rb")

animals = pickle.load(myFile)

print(animals)

myFile.close()
```

Y para poder guardar datos de indexación como un objeto clave-valor:

```
import shelve

myGodShelf = shelve.open("god.dat")

myGodShelf['name'] = "Azatoth"
myGodShelf['age'] = 13787
myGodShelf['immortal'] = True

myGodShelf.close()
```

Y leerlos de nuevo:

```
import shelve

myGodShelf = shelve.open("god.dat")

print(myGodShelf['name'])
print(myGodShelf['age'])
print(myGodShelf['immortal'])

myGodShelf.close()
```

Referencias

Manejo de excepciones
Gestor de paquetes
Módulos
Manejo de ficheros
Argumentos por línea de comandos