Gestión de datos y procesos en Python

Reading data interactively

Para leer datos de manera interactiva en Python, puedes usar la función input(). Esta función permite al usuario ingresar datos desde la línea de comandos.

```
# Leer datos del usuario
user_input = input("Ingrese su nombre: ")
print(f"Hola, {user_input}!")
```

Standard streams

Python maneja tres flujos estándar: stdin, stdout y stderr. Puedes redirigir estos flujos para leer datos de entrada, escribir salida y mostrar errores.

```
import sys

# Leer desde stdin
input_data = sys.stdin.read()
print("Datos recibidos:", input_data)

# Escribir en stdout
sys.stdout.write("Esto es una salida estándar.\n")

# Escribir en stderr
sys.stderr.write("Esto es un error estándar.\n")
```

Environment variables

Las variables de entorno en Python se pueden acceder usando el módulo os. Estas variables pueden contener información sobre el sistema operativo y la configuración del entorno.

```
import os

# Obtener una variable de entorno
home_dir = os.getenv('HOME')
print(f"Directorio home: {home_dir}")

# Establecer una variable de entorno
os.environ['MY_VAR'] = 'Valor de mi variable'
print(f"MY_VAR: {os.getenv('MY_VAR')}")
```

Command-line arguments and exit status

En Python, puedes usar el módulo sys para acceder a los argumentos de línea de comandos y el estado de salida del programa.

```
import sys

# Obtener argumentos de línea de comandos
args = sys.argv
print(f"Argumentos de línea de comandos: {args}")

# Establecer el estado de salida
sys.exit(0)  # Estado de salida 0 indica éxito
```

Running system commands in python

Puedes ejecutar comandos del sistema utilizando el módulo subprocess. Esto te permite interactuar con el sistema operativo y ejecutar comandos externos.

```
import subprocess

# Ejecutar un comando del sistema
subprocess.run(['ls', '-l'])  # En Linux
# subprocess.run(['dir'], shell=True)  # En Windows
```

Obtaining the output of a system command

Para capturar la salida de un comando del sistema, puedes usar el módulo subprocess y su función run() con el parámetro capture_output=True.

```
import subprocess

# Capturar la salida de un comando
result = subprocess.run(['ls', '-l'], capture_output=True, text=True)
print("Salida del comando:")
print(result.stdout)
```

Advanced subprocess management

Para una gestión avanzada de subprocesos, puedes usar subprocess.Popen para iniciar un proceso y comunicarse con él mediante sus flujos estándar.

```
import subprocess

# Iniciar un proceso con Popen
process = subprocess.Popen(['ping', 'localhost'], stdout=subprocess.PIPE, stderr=subprocess.PIPE)

# Leer la salida y errores
stdout, stderr = process.communicate()
print("Salida del proceso:")
print(stdout.decode())
```

```
print("Errores del proceso:")
print(stderr.decode())
```

What are log files?

Los archivos de registro (log files) son archivos que contienen registros de eventos y actividades del sistema, aplicaciones o servicios. Se utilizan para monitorear y depurar sistemas.

```
# Ejemplo de lectura de un archivo de log
with open('example.log', 'r') as file:
    log_contents = file.read()
    print("Contenido del archivo de log:")
    print(log_contents)
```

Filtering log files with regular expressions

Para filtrar archivos de log utilizando expresiones regulares, puedes usar el módulo re. Esto te permite buscar patrones específicos en el contenido del archivo.

```
import re

# Leer el archivo de log
with open('example.log', 'r') as file:
    log_contents = file.read()

# Filtrar líneas que contienen un patrón específico
pattern = re.compile(r'ERROR')
matches = pattern.findall(log_contents)

print("Errores encontrados en el archivo de log:")
for match in matches:
    print(match)
```