GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Unidad 1.2

Los procesos

Índice

Sumario

Licencia

1. INTRODUCCIÓN

Como hemos visto en el unidad 1.1 que un proceso es un programa que se está ejecutando en un sistema operativo concreto. Cada vez que ejecutamos una aplicación o un script en nuestro ordenador, se crea un proceso para gestionar esa ejecución. Los procesos son esenciales para el funcionamiento de cualquier sistema operativo, ya que permiten la multitarea, la gestión de recursos y la ejecución concurrente de múltiples tareas.

Python ofrece herramientas para interactuar con procesos y gestionarlos eficazmente. La biblioteca `subprocess` es una de las herramientas más útiles de Python para trabajar con procesos. Esta librería facilita la creación, manipulación y control de procesos. Con esta librería, se pueden por ello realizar una variedad de tareas relacionadas con procesos, como:

1. **Ejecutar programas externos:** Se utiliza para iniciar y ejecutar programas externos desde un script de Python. Esto es útil cuando se desea automatizar tareas que involucran la interacción con programas o herramientas externas.
2. **Comunicación con procesos:** Utilizando la librería podemos hacer que nuestro programa se comunique con procesos externos mediante la lectura y escritura de datos en sus flujos de entrada y salida. Esto es útil para interactuar con programas en tiempo real.
3. **Esperar a que los procesos terminen:** Se utiliza para esperar a que un proceso se complete antes de continuar con la ejecución del programa principal, lo que es esencial para la secuenciación de tareas y la gestión de dependencias.
4. **Manejo de errores y excepciones:** La librería `subprocess` maneja excepciones y errores relacionados con procesos de manera eficiente, lo que facilita la detección y el manejo de problemas en la ejecución de programas externos.

En resumen, la gestión de procesos es una parte esencial de la programación en sistemas operativos. Se utiliza sobre todo cuando se necesita automatizar tareas, interactuar con programas externos o realizar procesamiento en paralelo.

2. CREACIÓN DE PROCESOS

En Python, puedes utilizar la librería subprocess para crear y gestionar procesos. A continuación, se muestra un ejemplo de cómo lanzar un proceso para abrir un proceso calculadora. (*Con este programa se hace lo mismo que la acción de hacer doble click sobre el icono de la calculadora*).

Se usará el módulo de Python subprocess para lanzar los procesos. Se recomienda usar el método o función **run** , aunque otras funciones como **Popen** también sirven para éste propósito.

https://docs.python.org/es/3.10/library/subprocess.html

2.1. Abrir la calculadora o el notepad en Windows:

```
import subprocess

try:
    subprocess.run(['Notepad.exe'])
    subprocess.Popen(['C:\\Windows\\System32\\calc.exe'])
except FileNotFoundError:
    print("Adobe Acrobat Reader no encontrado en la ubicación predeterminada.")
except Exception as e:
    print("Se produjo un error al abrir Adobe Acrobat Reader:", str(e))
```

Se puede usar la ruta relativa o la ruta absoluta para abrir el ejecutable. He comentado las alternativas. Las puedes ir probando. Puedes combinar las dos opciones que propongo: run+ruta absoluta o Popen+ruta relativa. Te he propuesto como alternativa el Notepad, por si clac no funcionara.

2.2. Abrir la calculadora en una máquina Ubuntu:

Puede ser gnome-calculator o mate-calculator.

```
import subprocess

try:
    subprocess.run(['mate-calculator'])
    #subprocess.Popen(['mate-calculator'])
except FileNotFoundError:
    print("Programa calculadora no encontrado en la ubicación predeterminada.")
except Exception as e:
    print("Se produjo un error al abrir el programa calculadora:", str(e))
```

Puedes probar de abrir gnome-terminal o mate-terminal.

2.3. Abrir Adobe Acrobat Reader en macOS:

En macOS propongo probar de abrir Acrobat reader. El programa debe estar instalado. Revisando documentación, he visto que subprocess en macOs no va demasiado bien.

```
import subprocess

try:
    subprocess.Popen(['open', '-a', 'Adobe Acrobat Reader'])
except FileNotFoundError:
    print("Adobe Acrobat Reader no encontrado. Asegúrate de que esté instalado en tu sistema Mac.")
except Exception as e:
    print("Se produjo un error al abrir Adobe Acrobat Reader:", str(e))
```

Nota : Yo uso Ubuntu y Windows. Hay ejemplos de macOS que no están probados. Para las personas que estén usando macOS, si tienene muchos problemas con Visual Studio y Python, aconsejo **usar máquinas virtuales** (por ejemplo con Ubuntu) para realizar las prácticas.

2.4. Crear un proceso y pasar argumentos

Ahora vamos a pesarle parámetros proceso que abrimos. En este ocasión abriremos el proceso ping y le pasaremos los parámetros que deseamos. Queremos hacer 2 ping a una página web.

Para Linux/Ubuntu:

```
import subprocess

try:
    subprocess.run(["ping","www.fsf.org","-c","2"])
except subprocess.CalledProcessError as e:
    print(e.output)
```

Para Windows:

```
import subprocess

try:
    subprocess.run(["ping","www.fsf.org","-n","2"])
except subprocess.CalledProcessError as e:
    print(e.output)
```

Propongo que hagas este ejercicio:

Lanza un proceso que abra el navegador con la búsqueda en Google de un término solicitado al usuario.

Nota: Necesitaras concatenar la búsqueda. Abre el navegador y mira dónde se pone lo que buscas en la url del navegador.

3. OPERACIONES CON FICHEROS

Para explicar la comunicación entre procesos usaremos la comunicación entre ficheros. Tengo un documento de texto que contiene una información que usa mi programa para transformar esos datos y enviar el resultado a otro fichero. Es muy importante entender que todas estas operaciones se ejecutan en **un único proceso**.

Más adelante, en el siguiente tema, sí que ejecutaremos este programa varias veces con threads, y por tanto habrán varios hilos activo y ejecutándose en el mismo momento.

Ejercicio base:

Supongamos que necesitamos crear un programa que aproveche al máximo el número de CPUs para realizar alguna tarea intensiva. Supongamos que dicha tarea consiste en sumar números. En concreto le vamos a programar un programa en Python que sea capaz de sumar todos los números comprendidos entre dos valores incluyendo ambos valores.

Para resolver este problema, lo vamos a abordar por fases. Primero crearemos el programa que sea capaz de sumar todos los números comprendidos entre dos valores incluyendo ambos valores.

```
resultado = 0
print("Este programa suma todos los números comprendidos entre dos valores.")
n1 = int(input("Indica el primer valor: "))
n2 = int(input("Indica el segundo valor: "))
for i in range(n1, n2 + 1):
    resultado += i
print("La suma de los números entre", n1, "y", n2, "es:", resultado)
```

```
Este programa suma todos los números comprendidos entre dos valores.
Indica el primer valor: 1
Indica el segundo valor: 2
La suma de los números entre 1 y 2 es: 3
```

numeros.txt

```
1,2
5,7
3,5
```

Ahora lo que haremos es crear un fichero txt en el cual guardamos varias iteraciones del programa. Proponemos hasta 3 pares de números:

Le diremos a nuestro programa que lea los pares de valores. Luego para cada pareja de valores haga la operación y escriba en otro fichero el resultado.

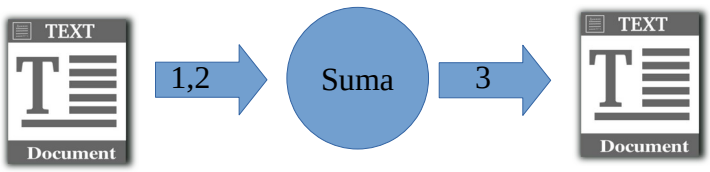

Imagen: **Shared By:** OCAL 05-09-2011

El programa propuesto sería:

```python
def sumar(n1, n2):
    """Dados dos números, la función retorna la suma de todos los números comprendidos entre n1 y n2"""
    resultado = 0
    for i in range(n1, n2 + 1):
        resultado += i
    return resultado

if __name__ == "__main__":

    resultado = 0

    #para abrir el fichero que se encuentra en el mismo path que el programa python
    fichero_path = Path(__file__).parent
    try:
        f1 = open(fichero_path/"solucion.txt","w")
        #La opción w crea el fichero, si no existe, y si existe sobrescrbe el contenido
    except OSError as e:
        print(f"Fichero1: ",e)

    else:
        try:
            f2 = open(fichero_path/"numeros.txt","r")
            #La opción r abre un ficer en formato sólo lectura
            for line in f2.readlines():
                n1, n2 = [int(x) for x in line.split(",")]
                resultado = sumar(n1,n2)
                print("La suma de los números entre", n1, "y", n2, "es:", resultado, file=f1)
        except OSError as e:
            print(f"Fichero2: ",e)
```

numeros.txt

```
1 1,2
2 5,7
3 3,5
```

solucion.txt

```
1 La suma de los números entre 1 y 2 es: 3
2 La suma de los números entre 5 y 7 es: 18
3 La suma de los números entre 3 y 5 es: 12
```

Nota: ¡**Usa TRY o WITH para controlar los errores**! Y cuidado con reaprovechar código de otras personas, sin conocer bien lo que hace.

La explosión del cohete Arian 5:
https://www.youtube.com/watch?v=5tJPXYA0Nec

Imagen: Spotting973, CC BY-SA 2.0 <https://creativecommons.org/licenses/by-sa/2.0>, via Wikimedia Commons

4. Anexo

4.1. Anexo 1: if __name__ == "__main__":

La expresión **if __name__ == "__main__"**:

Es una **convención** importante que se utiliza para determinar si el archivo Python se está ejecutando como un programa principal o si se está importando como un módulo en otro programa.

En Python, cuando ejecutas un archivo, se establece una variable especial llamada __name__ en el entorno global. Si el archivo se ejecuta directamente (programa principal), el valor de __name__ se establece en "__main__". Si el archivo se importa como un módulo, el valor de __name__ será el nombre del módulo.

Dicho de otra forma, si quiero reaprovechar funciones que han sido definido en otros programas de Python y usarlas en un nuevo programa, lo hacemos mediante la importación de archivos (como importar clases en java).

Cuando importo un programa, se importa todo, aunque a mi sólo me interese ciertas partes (las funciones por ejemplo). Es por eso que hay 2 “main” en mi nuevo programa:

- El main del programa que importo (ejemplo: **sumar**.py) – *(No se ve)*
- El main del programa que estoy creando (ejemplo: **operaciones_complejas**.py)

Python establece la etiqueta `__name__` = "main" para **el programa principal** que se está ejecutando (ejemplo: Para **operaciones_complejas**.py su __name__ sería igual a “main”).

En cambio el main del programa que **he importado** (ejemplo: **sumar**.py) no asigna a la etiqueta `__name__` ≠ "main" (diferente). En su lugar, pone el nombre real del archivo (en este ejemplo para **sumar.py** su __name__ sería igual a “suma”).

De esta forma la expresión if __name__ == "__main__": encapsula el código que deseas que se ejecute solo cuando el archivo se ejecuta directamente como un programa independiente (principal), y no cuando se importa como un módulo en otro programa.

Esto es especialmente útil en proyectos más grandes donde tienes varios archivos Python que pueden servir como scripts o módulos reutilizables.