61TE22_02_Automatización en Python

El **proceso de automatización** en Python se refiere a la secuencia de pasos o fases por las cuales se implementa un sistema que ejecuta tareas repetitivas o complejas de manera automática. A grandes rasgos, este proceso sigue un flujo estructurado que garantiza que las tareas se lleven a cabo sin intervención humana.

A continuación, te explico las etapas comunes del **proceso de automatización** usando Python:

1. Identificación de tareas repetitivas

Lo primero en un proceso de automatización es identificar qué tareas deben automatizarse. Estas suelen ser actividades que consumen tiempo o que se realizan con frecuencia, como:

- Procesamiento de grandes volúmenes de datos.
- Mantenimiento de archivos.
- Extracción y procesamiento de información desde la web.
- Generación de reportes o análisis de datos.
- Envío de correos o notificaciones.

Ejemplo: Imagina que tienes que descargar diariamente un informe de ventas de una plataforma web y generar gráficos de tendencias. Este tipo de tarea es un buen candidato para ser automatizado.

2. Diseño del flujo de automatización

Después de identificar las tareas, debes diseñar cómo se realizará la automatización. Esto incluye:

- Determinar **los pasos específicos** que tomaría una persona para completar la tarea.
- Decidir qué **herramientas y librerías** usarás para completar cada paso. En Python, estas herramientas podrían incluir:
    - **Pandas** para el manejo de datos.
    - **Selenium** para interactuar con sitios web.
    - **Schedule** para programar la ejecución en momentos específicos.

Ejemplo: Si tu tarea es descargar un archivo y procesarlo, el flujo podría ser:

1. Abrir una página web.
2. Navegar a la sección de informes.
3. Descargar el archivo.
4. Procesar el archivo con Python para generar gráficos.

61TE22_02_Automatización en Python

3. Desarrollo del script de automatización

Una vez tienes claro el flujo, es el momento de **escribir el código** en Python que hará todo el trabajo por ti. Aquí es donde seleccionas y usas las librerías que permitirán completar cada paso de la tarea.

Ejemplo: Si el flujo incluye descargar un archivo y generar un gráfico, el script en Python podría verse como algo así:

```
import requests
import pandas as pd
import matplotlib.pyplot as plt

# Descargar archivo desde la web
url = 'https://example.com/informe_ventas.csv'
response = requests.get(url)

# Guardar archivo localmente
with open('informe_ventas.csv', 'wb') as file:
    file.write(response.content)

# Leer y procesar el archivo
df = pd.read_csv('informe_ventas.csv')
df.plot(kind='line', x='Fecha', y='Ventas')
plt.savefig('grafico_ventas.png')
```

En este caso, el script descarga un informe de ventas desde una URL, lo guarda localmente y genera un gráfico con los datos.

4. Prueba y depuración

Es importante **probar el script** para asegurarse de que funciona como se espera en diferentes situaciones. Debes simular diferentes condiciones de trabajo:

- Probar con diferentes entradas.
- Manejar posibles errores o fallos en la red.
- Asegurarse de que el código pueda ejecutarse repetidamente sin problemas.

Ejemplo: En el script anterior, podrías probar cómo manejar el caso en el que la página web no esté disponible, o cuando el archivo no se descargue correctamente.

61TE22_02_Automatización en Python

5. Programación de la ejecución (si es necesario)

Una vez que el script esté completo y funcionando correctamente, puedes decidir **programar su ejecución** en intervalos regulares o en momentos específicos. Para esto, puedes usar herramientas como **Schedule** en Python o integrar tu script con **tareas programadas del sistema operativo** (como el **cron** en Linux o el **Task Scheduler** en Windows).

Ejemplo: Usar la librería schedule para ejecutar el script todos los días a las 9:00 AM:

```
import schedule
import time

def ejecutar_script():
    # Aquí puedes llamar tu función de automatización
    print("Ejecutando script de automatización...")

# Programar la ejecución
schedule.every().day.at("09:00").do(ejecutar_script)

while True:
    schedule.run_pending()
    time.sleep(60)  # Esperar un minuto entre chequeos
```

6. Monitoreo y ajustes

Incluso después de que el sistema de automatización esté funcionando correctamente, es crucial **monitorear** su desempeño. Los scripts pueden fallar debido a cambios en las páginas web (en caso de web scraping), archivos corruptos, o cualquier otro imprevisto.

Por lo tanto, debes asegurarte de:

- Implementar **mecanismos de manejo de errores**.
- Realizar ajustes periódicos para mantener la automatización actualizada.
- Integrar **logs** o registros para saber cuándo y por qué el sistema ha fallado.

Ejemplo: En el caso de descarga de archivos, podrías querer agregar un manejo de excepciones para registrar cualquier error en la descarga:

```python
try:
    response = requests.get(url)
    response.raise_for_status()  # Esto lanzará un error si la respuesta no es correcta
except requests.exceptions.RequestException as e:
    print(f"Error en la descarga: {e}")
```

7. Optimización

Finalmente, es importante optimizar la automatización para que consuma menos recursos y se ejecute de manera más eficiente. Esto puede incluir:

- Optimización de tiempos de ejecución.
- Mejor uso de la memoria.
- Minimizar interacciones innecesarias (especialmente en tareas de red o de web scraping).

61TE22_02_Automatización en Python

Componentes clave de la automatización con Python:

1. **Scripts de Python**: Son los programas o conjuntos de instrucciones que se ejecutan de forma automática.
2. **Tareas repetitivas**: Son aquellas acciones que se realizan una y otra vez, como mover archivos, analizar datos o llenar formularios.
3. **Librerías**: Python cuenta con numerosas librerías para automatizar tareas específicas, como:
    - **Pandas** para manipulación de datos.
    - **PyAutoGUI** para automatizar la interacción con interfaces gráficas.
    - **Selenium** para automatizar la navegación web.
    - **PyWin32** para interactuar con aplicaciones de Windows.
    - **schedule** para programar la ejecución de scripts.

Ejemplos de procesos de automatización en Python:

1. Automatización de archivos y directorios

Python se puede usar para organizar archivos automáticamente, mover archivos según su tipo, o realizar copias de seguridad.

```python
import os
import shutil

# Definir el directorio de origen y destino
source_dir = 'C:/Users/Usuario/Descargas'
dest_dir = 'C:/Users/Usuario/Documents/Backups'

# Crear el directorio de destino si no existe
if not os.path.exists(dest_dir):
    os.makedirs(dest_dir)

# Mover archivos desde el directorio de origen al de destino
for filename in os.listdir(source_dir):
    full_file_path = os.path.join(source_dir, filename)
    shutil.move(full_file_path, dest_dir)

print("Archivos movidos con éxito")
```

Este script mueve archivos de la carpeta "Descargas" a una carpeta de "Backups" de manera automática.

61TE22_02_Automatización en Python

2. Automatización de la extracción de datos de la web (web scraping)

Con Python, puedes automatizar la recolección de datos desde páginas web. Esto es útil cuando necesitas extraer datos de manera regular o procesar grandes cantidades de información de diferentes páginas.

```
import requests
from bs4 import BeautifulSoup

# Definir la URL de la página web a extraer datos
url = 'https://example.com'

# Hacer la solicitud HTTP a la página
response = requests.get(url)

# Crear el objeto BeautifulSoup para analizar la página
soup = BeautifulSoup(response.content, 'html.parser')

# Extraer todos los títulos de la página
titulos = soup.find_all('h2')

# Mostrar los títulos encontrados
for titulo in titulos:
    print(titulo.text)
```

Este script descarga una página web y extrae los títulos dentro de etiquetas <h2>, un uso típico de la automatización para recolectar información.

3. Automatización de reportes diarios

Puedes programar scripts que generen reportes automáticos basados en los datos disponibles en archivos o bases de datos.

```
import pandas as pd
from datetime import datetime

# Leer un archivo Excel con datos de ventas
df = pd.read_excel('ventas_diarias.xlsx')

# Filtrar las ventas de hoy
```

61TE22_02_Automatización en Python

```python
hoy = datetime.today().date()
ventas_hoy = df[df['Fecha'] == hoy]

# Calcular el total de ventas
total_ventas_hoy = ventas_hoy['Monto'].sum()

# Generar un reporte
reporte = f"Reporte de ventas del {hoy}\nTotal de ventas: {total_ventas_hoy}"

# Guardar el reporte en un archivo de texto
with open(f'reporte_{hoy}.txt', 'w') as file:
    file.write(reporte)

print("Reporte generado con éxito")
```

Este script toma datos de ventas diarias de un archivo Excel y genera un reporte con el total de ventas de ese día.

4. Automatización de la ejecución de tareas en tiempos programados

Python puede ser usado para ejecutar tareas a intervalos programados, por ejemplo, cada día a las 9 a.m.

```python
import schedule
import time

def tarea():
    print("Ejecutando tarea programada")

# Programar la tarea para que se ejecute todos los días a las 9:00 AM
schedule.every().day.at("09:00").do(tarea)

while True:
    # Comprobar si hay alguna tarea pendiente para ejecutar
    schedule.run_pending()
    time.sleep(1)
```

Este script usa la librería schedule para ejecutar una tarea todos los días a las 9:00 AM. Puede ser usado para automatizar la generación de informes, envío de correos, o cualquier otra tarea recurrente.

Tipos de procesos que se pueden automatizar con Python:

1. **Manipulación de archivos**: Renombrar, mover, copiar, eliminar archivos de manera automatizada.
2. **Procesamiento de datos**: Análisis de grandes volúmenes de datos usando Pandas o NumPy.

3. **Interacción con sitios web**: Usar Selenium o requests para automatizar la navegación, extracción de datos o la realización de tareas repetitivas.
4. **Tareas en aplicaciones de escritorio**: Usar PyAutoGUI para simular el comportamiento del ratón y el teclado en aplicaciones sin APIs.
5. **Tareas de red**: Monitorear redes, enviar notificaciones o ejecutar scripts cuando se detectan eventos específicos.

Beneficios de la automatización con Python:

- **Ahorro de tiempo**: Las tareas que se ejecutan automáticamente pueden realizarse en minutos, cuando de otro modo llevarían horas.
- **Reducir errores humanos**: Al eliminar la intervención manual, se reduce la probabilidad de errores.
- **Escalabilidad**: Las soluciones automatizadas pueden escalar fácilmente a cientos o miles de tareas sin mayor esfuerzo.
- **Integración fácil**: Python puede integrarse fácilmente con otros sistemas y tecnologías, lo que lo hace muy versátil.

Conclusión:

El "procesador de automatización" en Python hace referencia a los **scripts** y **librerías** que se usan para ejecutar tareas automáticamente. Python es muy popular para esto debido a su simplicidad y la gran cantidad de librerías que permiten interactuar con archivos, aplicaciones, bases de datos y páginas web.

61TE22_02_Automatización en Python

1. Automatización de envío de correos electrónicos con PyWin32

Ejemplo: Enviar correos automáticamente desde Outlook

Si trabajas con Microsoft Outlook, puedes automatizar el envío de correos utilizando Python. Esto es útil para enviar notificaciones, recordatorios, o informes de manera regular.

```
import win32com.client as win32

# Crear una instancia de la aplicación Outlook
outlook = win32.Dispatch('outlook.application')

# Crear un nuevo correo
mail = outlook.CreateItem(0)

# Configurar los detalles del correo
mail.Subject = 'Este es un correo automático'
mail.Body = 'Hola, este correo ha sido enviado automáticamente usando Python.'
mail.To = 'destinatario@example.com'  # Reemplaza con el correo del destinatario

# Si necesitas agregar un archivo adjunto
# mail.Attachments.Add('C:\\ruta\\al\\archivo.txt')

# Enviar el correo
mail.Send()

print("Correo enviado con éxito")
```

Este script se ejecuta en un sistema con Microsoft Outlook instalado y envía correos automáticamente. Puedes modificar el asunto, cuerpo y destinatarios según tus necesidades. También puedes agregar archivos adjuntos.

61TE22_02_Automatización en Python

2. Manipulación de datos de Excel con Pandas

Ejemplo: Automatizar el procesamiento de datos de ventas en Excel

Si trabajas con grandes hojas de cálculo y necesitas automatizar el análisis o reporte de datos, **Pandas** es la herramienta ideal. Este ejemplo muestra cómo leer un archivo de Excel, realizar operaciones sobre los datos y guardar los resultados.

```
import pandas as pd

# Leer un archivo Excel
df = pd.read_excel('ventas.xlsx')

# Mostrar las primeras filas del archivo
print(df.head())

# Sumar los valores de la columna 'Ventas'
total_ventas = df['Ventas'].sum()

print(f"El total de ventas es: {total_ventas}")

# Añadir una nueva columna con el total de ventas
df['Total Ventas'] = total_ventas

# Guardar el archivo actualizado
df.to_excel('ventas_actualizadas.xlsx', index=False)
```

Este script realiza lo siguiente:

- Lee datos de un archivo Excel.
- Suma una columna específica.
- Añade esa suma como una nueva columna y guarda los cambios en un nuevo archivo Excel.

61TE22_02_Automatización en Python

3. Creación de ejecutables con PyInstaller

Ejemplo: Convertir un script Python en un ejecutable

Con **PyInstaller**, puedes convertir tus scripts de Python en archivos ejecutables (.exe), lo que permite que otras personas los utilicen sin tener que instalar Python.

Si tienes un script llamado mi_script.py, puedes convertirlo en un archivo ejecutable con el siguiente comando:

yinstaller --onefile mi_script.py

Este comando genera un archivo ejecutable que incluye todo lo necesario para que tu script funcione en cualquier máquina sin tener que instalar dependencias adicionales.

5. Automatización de navegación web con Selenium

Ejemplo: Extraer datos de una página web

Si necesitas interactuar con un sitio web para extraer datos, llenar formularios o navegar automáticamente, **Selenium** es una opción poderosa.

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time

# Crear un objeto de navegador (asegúrate de tener el driver correcto instalado)
driver = webdriver.Chrome(executable_path='C:/ruta/al/chromedriver.exe')

# Abrir una página web
driver.get('https://www.google.com')

# Buscar un término en Google
search_box = driver.find_element_by_name('q')  # El campo de búsqueda en Google tiene el
nombre 'q'
search_box.send_keys('Automatización con Python')
search_box.send_keys(Keys.RETURN)  # Simula presionar "Enter"

# Esperar 3 segundos para ver los resultados
time.sleep(3)

# Capturar una captura de pantalla de los resultados
driver.save_screenshot('resultados_google.png')

# Cerrar el navegador
driver.quit()
```

61TE22_02_Automatización en Python

Este script abre un navegador, navega a Google, busca un término, y toma una captura de pantalla de los resultados.

Conclusión: ¿Qué se puede automatizar?

- **Emails masivos**: Automatiza el envío de correos electrónicos con Outlook y PyWin32.
- **Procesamiento de datos**: Manipula y analiza grandes cantidades de datos en Excel con Pandas.
- **Interacción con la interfaz**: Simula acciones del ratón y teclado para interactuar con aplicaciones sin API usando PyAutoGUI.
- **Navegación web**: Automatiza tareas en sitios web, como iniciar sesión o extraer datos, con Selenium.
- **Ejecutables**: Distribuye tus scripts como aplicaciones independientes con PyInstaller.