🌐 Automating Real-World Tasks with Python – Cheatsheet

Módulo 2: Interacting with Web Services

🎯 Introducción

Este módulo enseña cómo **interactuar con APIs y servicios web** para obtener, enviar o procesar datos desde Python.
Incluye el uso de **HTTP requests**, **JSON**, **APIs REST** y automatización de tareas online.

🧰 Herramientas principales

Librería	Uso
`requests`	Realizar peticiones HTTP (GET, POST, PUT, DELETE)
`json`	Leer y escribir datos en formato JSON
`urllib`	Alternativa estándar para acceder a URLs
`xml.etree.ElementTree`	Procesamiento de XML (cuando la API devuelve XML)

Instalación de `requests`

```
pip install requests
```

🧠 Conceptos clave

🔹 Hacer una solicitud GET

```
import requests

respuesta = requests.get("https://api.ejemplo.com/data")
print(respuesta.status_code)   # 200 -> éxito
print(respuesta.text)          # contenido bruto
```

🔹 Hacer una solicitud POST

```
data = {"nombre": "Juan", "edad": 30}
respuesta = requests.post("https://api.ejemplo.com/users", json=data)
```

```
print(respuesta.status_code)
```

Procesar JSON

```
import json

json_data = respuesta.json()  # devuelve dict de Python
print(json_data["resultado"])
```

Parámetros en la URL

```
params = {"q": "python", "limit": 5}
respuesta = requests.get("https://api.ejemplo.com/search", params=params)
```

Headers y autenticación

```
headers = {"Authorization": "Bearer TOKEN"}
respuesta = requests.get("https://api.ejemplo.com/protected", headers=headers)
```

Ejemplo de uso práctico

- Automatizar la descarga de datos de un servicio web:

```
import requests
import csv

url = "https://api.ejemplo.com/products"
respuesta = requests.get(url)
productos = respuesta.json()

with open("productos.csv", "w", newline="") as archivo:
    writer = csv.DictWriter(archivo, fieldnames=["id", "nombre", "precio"])
    writer.writeheader()
    for p in productos:
        writer.writerow(p)
```

> Con esto se puede automatizar la generación de reportes desde cualquier API REST.

Buenas prácticas

- Manejar errores y excepciones:

```
try:
    r = requests.get("https://api.ejemplo.com/data")
    r.raise_for_status()  # lanza error si status != 200
except requests.exceptions.RequestException as e:
    print("Error:", e)
```

- Usar **timeouts** para no bloquear scripts largos:

```
r = requests.get("https://api.ejemplo.com", timeout=10)
```

- Limitar la frecuencia de requests para respetar **rate limits** de la API.

Conceptos avanzados

- **Autenticación OAuth2** para APIs seguras.
- **Manejo de sesiones** con `requests.Session()` para mantener cookies.
- **Automatización de workflows** combinando múltiples endpoints.

Referencias

- Requests: HTTP for Humans
- Python JSON Module
- API Best Practices