ESTRUCTURAS DE CONTROL EN PYTHON

UT1: Preparación del entorno

GECEGS Desarrollo de Aplicaciones en Lenguaje Python
Curso 2025-26

Índice de contenidos

1. Comprobar e instalar Python

Para programar en Python, es necesario tener este lenguaje instalado en nuestro equipo. En esta sección veremos cómo verificar si Python ya está instalado y, si no lo está, cómo instalarlo.

1.1 Comprobar si Python está instalado

Abra un terminal o una consola de comando:

- En Windows, busque en el menú de inicio.
- En macOS o GNU/Linux, abra la aplicación "Terminal".

Escriba el siguiente comando:

```
python --version
```

Si Python está instalado, verás un mensaje con la versión. Si ves un mensaje de error o dice que el comando no es reconocido, entonces necesitas instalar Python.

1.2 Instalar Python

Si no tienes Python instalado o necesitas actualizarlo, puedes seguir los pasos apropiados de acuerdo con tu sistema operativo:

1. **Windows**: Accede al sitio web oficial: https://www.python.org/downloads/windows/ ; Haz clic en el botón de descarga para la última versión. Ejecuta el archivo descargado. En la ventana del instalador: Comprueba la opción **"Agregar Python a** PATH". Haz clic **en "Instalar ahora".**
2. **macOS**: Descarga el instalador desde: https://www.python.org/downloads/macos/ ; Abre el archivo .pkg descargado y sigue los pasos del instalador.
3. **GNU/Linux**: La mayoría de las distribuciones GNU/ Linux ya incluyen Python. Si no está instalado o quieres actualizarlo, puedes utilizar el administrador de paquetes correspondiente. Por ejemplo, en Debian/Ubuntu: 'sudo apt install python3'.

2. El gestor de paquetes *pip*

pip es una herramienta que permite instalar paquetes y bibliotecas adicionales en Python. Generalmente se instala automáticamente junto con Python. Verificar si ya está instalado con el comando:

```
pip --version
```

Si ves un mensaje con la versión de pip, entonces ya está instalado. Si aparece un mensaje de error que indica que el comando no es reconocido, tendrás que instalar pip manualmente:

1. **Windows o macOS**: Descarga el guión 'get-pip.py' del sitio oficial: https://bootstrap.pypa.io/get-pip.py ; Guarda el archivo en una ubicación fácil de acceder, por ejemplo en el escritorio. Abre una terminal o consola, navega a la carpeta donde se guarda el archivo y ejecuta: 'python get-pip.py'.
2. **GNU/Linux**: En la mayoría de las distribuciones GNU/ Linux se puede instalar pip con el gestor de paquetes. Por ejemplo, en Debian/Ubuntu: 'sudo apt install python3-pip'.

Una vez instalado pip, puedes utilizarlo para instalar paquetes de Python, tales como:

```
pip install nombre_de_paquete
```

Por el momento, no utilices el instalador pip. Espera a ver el siguiente punto para crear antes un entorno virtual.

3. Crear entornos virtuales

Un entorno virtual de Python es un espacio aislado en el que podemos instalar dependencias y paquetes específicos para un proyecto sin afectar a otras configuraciones del sistema.

Para evitar problemas con las dependencias entre proyectos y tener un entorno aislado para cada uno de ellos, es recomendable utilizar entornos virtuales. Además, en GNU/Linux, si no configuras un entorno virtual, tendrás problemas con pip. En Debian/Ubuntu puedes instalar la herramienta 'venv' con el comando 'sudo apt install python3-venv'.

Para crear un entorno virtual, abre una terminal o una consola de comando y desplázate a la carpeta donde deseas crear el entorno virtual. Ejecuta el comando siguiente para crear el entorno virtual (que aquí llamaremos 'mientorno'):

```
python -m venv mientorno
```

Este comando creará una carpeta llamada 'mientorno' que contendrá el entorno virtual.

Hemos de activar el entorno virtual a continuación. Para ello, en Windows invocaremos a:

```
mientorno\Scripts\activate
```

En GNU/Linux o macOS:

```
source mientorno/bin/activate
```

Después de activar el entorno virtual, debes ver el nombre del entorno (por ejemplo, 'mientorno') al principio de la línea en la consola, indicando así que está activo. Una vez activado el entorno, asegúrate de que el pip está disponible: 'pip –version'. A partir de ahí, los paquetes se instalarán **dentro del entorno virtual**.

Cuando termines de trabajar en su proyecto, puedes dejar el entorno virtual ejecutando:

```
deactivate
```

Esto te devolverá al entorno global del sistema.

4. Instalar paquetes con *pip*

En Python, los paquetes son módulos reutilizables que amplían la funcionalidad del lenguaje. Para administrar e instalar estos paquetes, se utiliza el gestor oficial de paquetes de Python.

Para instalar un paquete, utilice el siguiente comando en la consola (**preferiblemente con el entorno virtual habilitado)**:

```
pip install nombre_de_paquete
```

Para ver qué paquetes están instalados en tu entorno actual puedes utilizar:

```
pip list
```

Si necesitas **desinstalar** un paquete:

```
pip uninstall nombre_de_paquete
```

Para **actualizar** un paquete a su última versión:

```
pip install --upgrade nombre_de_paquete
```

También es posible instalar varios paquetes listados en un archivo 'requirements.txt' con la orden siguiente:

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

Este archivo contiene una lista de paquetes, uno por línea, que el proyecto necesita.

Puedes generar el archivo 'requirements.txt' de tu proyecto con la orden:

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

Pero, si quieres que 'requirements.txt' incluya solo las dependencias que realmente se usan en tu proyecto (no todo lo que está instalado en el entorno virtual), puede utilizar herramientas como 'pipreqs':

```
pip install pipreqs
pipreqs /ruta/a/tu/proyecto
```

Esto creará un archivo requirements.txt solo con los paquetes utilizados en el código fuente. El fichero 'requirements.txt' es importante porque:

- Contiene todas las dependencias de tu proyecto (con las versiones). Esto permite a otras personas (o a ti mismo más tarde) instalar exactamente las mismas bibliotecas y versiones.
- Cuando trabajas en equipo, los otros desarrolladores pueden preparar su entorno con un solo comando. Es una práctica habitual en proyectos Python.
- Plataformas como Heroku, AWS, Docker, etc. usan 'requirements.txt' para instalar dependencias cuando implementas una aplicación.
- Permite asegurarse de que las versiones de los paquetes que se están utilizando no cambian inesperadamente. Esto evita errores cuando se publican nuevas versiones incompatibles.
- Es parte del estándar de cualquier proyecto Python bien estructurado (junto con otros archivos como 'README.md', '.gitignore', etc.).

5. Instalar un IDE

Para escribir y ejecutar código en Python de manera eficiente, es recomendable utilizar **un editor de texto avanzado** o **un IDE (Entorno de Desarrollo Integrado).** Estas herramientas ofrecen características como resaltado de sintaxis, autocompletado, depuración y gestión de proyectos.

1. Visual Studio Code es uno de los editores más populares para el desarrollo de Python. Es ligero, extensible y libre.

- Descárgalo desde: https://code.visualstudio.com/

- Instálalo como cualquier otra aplicación para su sistema operativo.
- Una vez abierto, instala la extensión oficial de Python:
    - Haz clic en el icono de extensiones (panel lateral izquierdo).
    - Haz una búsqueda: Python (editor Microsoft) e instálala.
    - Instala otras extensiones para Python de Microsoft: Python Debugger, Pylance, Entornos Python, etc.

Con estas extensiones instaladas, puedes ejecutar scripts, usar depuración y trabajar con entornos virtuales de VS Code. La depuración es el proceso de encontrar y solucionar errores en el código fuente de cualquier software. Para depurar tu código Python, puedes usar herramientas como pdb, que es el depurador integrado Python. Sólo tienes que incluir la línea: **import pdb; pdb.set_trace()** a su código para empezar a depurar. Sin embargo, utilizar el depurador integrado en VS Code, más visual e intuitivo.

2. PyCharm ofrece un potente autocompletado de código, depuración integrada, integración con bases de datos, refactorización o soporte para frameworks web como Django o Flask. Puedes descargarlo desde https://www.jetbrains.com/pycharm/download ; Ejecuta el instalador y sigue las instrucciones. Configura el intérprete de Python desde el IDE tras la instalación.

3. Jupyter Notebook es ideal para el análisis de datos. Ofrece una web interactiva que permite escribir código, texto o ecuaciones en un mismo documento. Puedes instalarlo, instalando primero Anaconda (https://www.anaconda.com/download/success) o instalando solo Jupyter con 'pip install notebook'. Para ejecutarlo, invócalo desde la terminal con 'jupyter notebook' y abre un navegador para acceder al entorno (http://localhost:8888).

4. Spyder es un IDE muy usado en ciencia de datos. Ofrece integración con IPython, visualización de datos y autocompletado de código. Para instalarlo, puedes instalar primero Anaconda o instalar solo Spyder con 'pip install spyder'. Abre Spyder desde el menú de aplicaciones o usando 'spyder' en la terminal.

6. Software para el control de versiones

El control de versiones te permite gestionar los cambios en el código fuente de manera eficiente. La herramienta más utilizada para esto es **Git**, que permite mantener un historial de versiones, colaborar con otras personas y trabajar en múltiples ramas (versiones) del proyecto.

Puedes descargar Git desde su sitio oficial: https://git-scm.com/ ; La instalación es simple y depende del sistema operativo. En Windows, incluye una terminal llamada **Git Bash**.

Después de instalar Git, es recomendable configurar la herramienta estableciendo tu nombre de usuario y correo electrónico:

```
git config --global user.name "Tu nombre"
git config --global user.email "tu.email@ejemplo.com"
```

6.1 Órdenes básicas de *git*

Inicializar un repositorio:

```
git init
```

Ver el estado del repositorio:

```
git status
```

Añadir un archivo al área de preparación:

```
git add nombre_de_archivo.py
```

O bien para añadir todos los archivos que han sido modificados:

```
git add . --all
```

Hacer una confirmación (guardar los cambios):

```
git commit -m "Mensaje descriptivo del cambio"
```

Ver el historial de confirmaciones:

```
git log
```

6.2 Uso de *GitHub* como repositorio remoto

GitHub es una plataforma para albergar repositorios de Git en la nube. Te permite colaborar y respaldar proyectos. Para emplearlo:

1. Crea una cuenta en https://github.com/
2. Crea un nuevo repositorio desde la web.
3. Conecta tu repositorio local con GitHub con:

```
git branch -M main
git remote add origin https://github.com/usuario/nombre-repositorio.git
git push -u origin main
```

Estos comandos solo serán necesarios una vez. Realizada la conexión, si queremos sincronizar nuestro repositorio local con el remoto, después de una confirmación, simplemente usaremos:

```
git push
```

7. Configuración de entornos seguros

Cuando se trabaja en proyectos de software, es importante mantener un entorno seguro para proteger las credenciales, datos sensibles y evitar errores debido al mal manejo de la configuración. A continuación se presentan buenas prácticas para lograrlo.

7.1 Variables de entorno

Las variables de entorno permiten almacenar información sensible, como claves API o contraseñas, sin incluirlas directamente en el código fuente. Para gestionar estas variables, puedes utilizar un archivo .env y necesitarás la biblioteca python-dotenv:

Ejemplo de archivo .env:

```
SECRET_KEY = tu_clave_secreta
DEBUG = False
```

Y para acceder a la clave desde tu aplicación Python:

```
from dotenv import load_dotenv
import os
load_dotenv()
clave = os.getenv("SECRET_KEY")
```

7.2 Utilización del archivo *.gitignore*

El archivo .gitignore le indica a Git qué archivos o carpetas no deben incluirse en el control de versiones. Es importante asegurarse de que, por ejemplo, el contenido de los entornos virtuales, los archivos de tipo .env, los archivos temporales, los archivos de configuración locales, etc. no se suban a repositorios públicos.

Ejemplo de archivo .gitignore:

```
venv/
__pycache__/
*.pyc
.env
```

7.3 Gestión de dependencias seguras

- Utiliza entornos virtuales para aislar tus proyectos.
- Congela las versiones de tus paquetes con:

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

- Revisa actualizaciones y vulnerabilidades en las dependencias con herramientas como 'pip-audit':

```
pip install pip-audit
pip-audit
```

7.4 Control de acceso y buenas prácticas

En entornos de colaboración o producción es especialmente importante:

- No compartir credenciales por correo electrónico o en repositorios.
- Utilizar autenticación de dos pasos en plataformas como GitHub.
- Aplicar principios de privilegio mínimo a tus entornos.

8. Bibliotecas

Python tiene un gran número de bibliotecas con diferentes propósitos que los programadores pueden utilizar. Entre las más importantes y populares podemos mencionar

- os / sys
- datetime
- logging
- argparse
- tkinter

La biblioteca TkInter es una de las más populares y completas en Python para la creación de interfaces gráficas de usuario (GUI). Nos permitirá desarrollar aplicaciones con ventanas, botones, etiquetas, cajas de texto, listas o menús.

Ver: https://docs.python.org/3/library/tkinter.html

Ejemplo de uso:

```
import tkinter as tk
ventana = tk.Tk()
ventana.title("Mi primera GUI")
ventana.geometry("300x100")

etiqueta = tk.Label(ventana, text="¡Hola, mundo!")
etiqueta.pack()

ventana.mainloop()
```

Algunas bibliotecas no están incorporadas por defecto en el lenguaje Python y deben ser instaladas primero (en nuestro entorno virtual).

Otras bibliotecas agrupadas por temática:

Herramientas y utilidades generales

- **os / sys**: Módulos nativos para interactuar con el sistema operativo y los parámetros.
- **datetime**: Gestión de fechas y horas.
- **logging**: Registro de actividad para depuración y mantenimiento.
- **Argparse**: Análisis de argumentos pasados por línea de comandos.
- **pathlib**: Manipulación de rutas de archivo de una manera moderna.

Interfaces de usuario gráficas (GUI)

- **Tkinter**: Biblioteca de Python estándar para crear interfaces gráficas fácilmente.
- **PyQt / PySide**: Interfaces modernas y potentes basadas en el framework Qt.
- **Kivy**: Diseñado para crear aplicaciones con interfaz táctil, también útil para móviles.
- **wxPython**: GUI nativo multiplataforma con controles avanzados.

Web y APIs

- **Flask**: Micro-framework para aplicaciones web ligeras.
- **Django**: Framework completo para el desarrollo web con bases de datos, autenticación, etc.
- **FastAPI**: Desarrollo muy rápido de APIs.
- **Requests**: Envío de peticiones HTTP sencillas e intuitivas.
- **BeautifulSoup**: Extracción de datos de páginas web (web scrapping).
- **Selenium**: Automatización y pruebas de interfaz en navegadores web.

Bases de datos

- **SQLAlchemy**: ORM para trabajar con bases de datos SQL.

- **SQLite3**: Incluido con Python, ideal para bases de datos ligeras.
- **Peewee**: ORM sencillo y ligero.
- **PyMongo**: Conexión con bases de datos MongoDB (NoSQL).

Ciencia de datos y análisis numérico

- **NumPy**: Tratamiento eficiente de vectores, matrices y cálculos matemáticos.
- **Pandas**: Manipulación y análisis de datos estructurados (tablas, series, etc.).
- **Matplotlib**: Creación de gráficos y visualizaciones 2D.
- **Seaborn**: Visualizaciones estadísticas más elegantes basadas en Matplotlib.
- **SciPy**: Funciones científicas avanzadas (álgebra lineal, estadística, cálculo numérico).
- **Plotly**: Gráficos interactivos para la web y el análisis exploratorio.

Ciencia e ingeniería

- **SymPy**: Cálculo simbólico (matemáticas simbólicas).
- **OpenCV**: Procesamiento de imágenes y visión artificial.
- **Biopython**: Bioinformática y análisis de datos biológicos.
- **Astropy**: Astronomía y manipulación de datos astronómicos.

9. Otros entornos de programación para Python

1. Python Shell es el modo interactivo básico de Python. Cuando instalas Python, viene con un entorno de consola interactiva que te permite ejecutar código Python línea a línea y obtener resultados inmediatos. Para usarla, en una terminal (cmd o PowerShell en Windows), simplemente escribe 'python'. Puedes salir con 'exit'.

2. IDLE (Entorno de Desarrollo Integrado para Python) es un entorno gráfico de desarrollo elemental que permite editar y ejecutar programas en Python. IDLE también es un entorno interactivo en el que se puede ejecutar instrucciones sueltas de Python. En Windows, IDLE se distribuye junto con el intérprete de Python (es decir, en su cuando instalas Python en Windows, IDLE también se instala automáticamente). En Linux, IDLE se distribuye como una aplicación separada que se puede instalar desde los repositorios de cada distribución.

3. Python desde Bash: Puedes usar un editor de texto de tu sistema operativo (nano, gedit, notepad en Windows, VSCode, etc.), creamos un archivo con extensión '.py' con nuestro código y podemos ejecutarlo desde una terminal.

Por ejemplo, si creamos un archivo llamado 'ejemplo.py':

```
a = 5
b = 3
suma = a + b
print("La suma de", a, "y", b, "es:", suma)
```

...guardamos el archivo, y ejecutamos el código en la terminal con:

```
python ejemplo.py
```

Otra forma de ejecutar un archivo Python es indicando en la primera línea donde se encuentra el intérprete de Python:

```
#! /usr/bin/python3
nombre = input("Hola, dime tu nombre: ");
print("¡Hola ", nombre , "practica código en Python!");
```

...guardamos el fichero, y ejecutamos el código en la terminal con:

```
$ chmod +x ejemplo.py
$ ./ejemplo.py
```

4. IPython es una versión mejorada de Python Shell que proporciona funcionalidades adicionales como autocompletado (con la tecla tabulador), documentación en línea (nombre del comando seguido de '?'), historial de comandos (flechas arriba y abajo) y más. Para usarlo, después de la instalación, simplemente escribe 'ipython' desde la terminal.

5. Python interactivo desde scripts: Si prefieres escribir tu código en archivos '.py', pero quieres ejecutar interactivamente ciertas secciones, puedes utilizar la función 'code.InteractiveConsole' para emular un entorno interactivo dentro de un guion de Python.

Ejemplo de uso de InteractiveConsole:

```
import code

# Crear una consola interactiva
console = code.InteractiveConsole()

# Ejecutar código en la consola
console.push('x = 10')
console.push('print(x * 2)')
```

Este enfoque es útil si se desea incorporar una sesión interactiva en una aplicación o script Python (por ejemplo, para pruebas, prototipos o entornos educativos).

6. Python desde plataforma web: Hay plataformas web desde las que se puede practicar código Python tecleando directamente en el navegador:

- **Kaggle**: https://www.kaggle.com/
- **Google Colab**: https://colab.research.google.com/
- **Replit**: https://replit.com/languages/python3
- **Programiz**: https://www.programiz.com/python-programming/online-compiler/
- **Online Python**: https://www.online-python.com/
- **W3Schools**: https://www.w3schools.com/python/