Versiones y Dependencias

TB022 - Esteban - Riesgo - Kristal

Manejo de Dependencias

Cada lenguaje de programación tiene una biblioteca estándar con funciones, clases, tipos, etc. útiles para el desarrollo de cualquier sistema.

Además de ellos, cada individuo o grupo de personas puede desarrollar sus propias dependencias con funcionalidad custom, para ser usadas en cualquier otro proyecto.

En python, estas dependencias se manejan usando pip

pip es una manejador de dependencias de Python (equivalente a lo que apt es para Ubuntu). Estas dependencias se almacenan en un repositorio https://pypi.org/ al cual cualquiera puede acceder.

En nuestros proyectos podemos usar estas dependencias si previamente las instalamos en nuestro sistema.

pip: Modo de uso

Para instalar una dependencia simplemente debemos hacer

```
pip install <dependencia>
```

Pero es buena práctica hacer lo siguiente:

Cada uno de nuestros proyectos tendra un archivo requirements.txt que va a especificar las dependencias que ese proyecto puntual va a necesitar.

Luego, en vez de instalar una por una, se instalarán todas juntas usando

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

De esta manera, trackeamos fácilmente qué dependencias necesitamos en cada proyecto, además de instalarlas también corriendo el comando una sola vez.

Manejador de versiones de Python

Toda distribución de Linux y macOS también, viene con una versión de Python instalada por defecto. La misma va a depender de la versión del SO o distribución que se use.

Hay ciertos procesos del SO que pueden depender de la versión de Python que esté instalada. En general no es recomendable cambiarla. Si queremos usar una versión distinta, más nueva o más vieja, podemos usar herramientas que me permiten instalar multiples versiones en un mismo sistema.

Generalmente esto se debe a que los proyectos se escriben para una versión particular del lenguaje. Con el tiempo, distintas versiones pueden tener cambios que hagan que el sistema se comporte diferente, o directamente no funcione. En estos casos, que en el mundo real, son casi todos, es que vamos a querer tener instaladas multiples versiones del lenguaje que usemos, y en cada proyecto usar una versión específica.

Para Python, vamos a usar pyenv.

pyenv: Modo de uso

Instalación de la herramienta y setup inicial

Ver el README publicado con instrucciones

```
pyenv install -l
```

Lista todas las versiones disponibles para instalar

```
pyenv versions
```

Lista las versiones instaladas

```
pyenv install <version>
```

Instala una versión particular

```
pyenv [local|global] <version>
```

Setea local o globalmente una version (tiene que haber sido instalada antes)

Hay que seleccionar desde VSCode (y desde otros IDEs probablemente también) qué versión de Python queremos usar para los plugins que tengamos allí también.

pyenv: Configuración para VSCode

Para hacer que VSCode tome la versión de python que queremos, tenemos que manualmente apuntar a ella.

Para ello necesitamos obtener el path de la versión que instalamos con pyenv. Probablemente sea `~/.pyenv/versions/<version>/bin/python`

Luego, en VSCode hacer click abajo a la derecha donde está la versión de Python actual, e ingresar el path de la versión.

Virtual Environments

El segundo problema que suele ocurrir, es que dos proyectos distintos pueden llegar a usar la misma versión del lenguaje, pero cada uno puede tener distintas dependencias. Pueden usar distintas bibliotecas, o hasta distintas versiones de las mismas bibliotecas.

Para ello, surgen los virtual environments. En cada proyecto se seteará un virtual env en el cual se instalarán todas las dependencias necesarias.

Para manejar virtual envs en Python, vamos a usar venv.

venv: Modo de uso

No es necesario instalar nada, la herramienta ya viene por defecto con Python.

En la terminal, debemos correr

```
python3 -m venv <dir>
```

Para crear un venv en el directorio actual. Se creará un directorio <dir> con las especificaciones del virtual env.

A continuación en cada sesión de la terminal que usemos debemos activar el virtual env

```
source <dir>/bin/activate
```

El venv también se puede crear desde VSCode directamente. Debemos seleccionar nuevamente el intérprete de Python a usar.

pipenv: Modo de uso

`pipenv lock`

Crea el Pipfile.lock

`pipenv --python <version>`

Especifica qué versión de python usar para este proyecto (de las que instalamos con pyenv)

`pipenv shell`

Activa el virtual environment

PEP8

PEP?

Python Enhancement Proposals. Son propuestas hechas por la comunidad sobre cómo mejorar el lenguaje.

PEP8 está escrito por Guido van Rossum (creador del lenguaje) junto con un par más de personas. Es una *guía de estilo* que se recomienda seguir para formattear código Python.

Todo código Python debería seguir esta guía. Todo código que se escriba en esta materia va a tener que seguirla.

Pero hacer estos cambios manualmente puede ser molesto y costoso..

Linter

Un Linter es una herramienta que permite detectar en el código desviaciones de las guías de estilo especificadas, y hasta arreglar esas diferencias automáticamente.

Un linter es una herramienta de *análisis estático de código*

Algunos lenguajes ya lo tienen incorporado por defecto. Python por ejemplo, no. Existen bibliotecas que contienen esta funcionalidad. Particularmente para este lenguaje hay muchísimas opciones.

Nosotros vamos a usar *black*

Pylance

Herramienta para chequear por posibles "errores" en el código, que tenga el formato que establece la guía de estilo, entre otras cosas.

Instalar

```
pip install pylint
```

Correr

```
pylint <path_al_archivo_o_dir>
```

El resultado es un puntaje de 0 a 10 y un listado de cosas a mejorar.

La herramienta es integrable a VSCode. Para ello hay que descargarse el plugin. Correrá automáticamente al abrir un archivo .py

black

Linter y auto formatter. Permite adaptar el código ya escrito a los estándares establecidos.

Instalar

```
pip install black
```

Correr

```
black <path_al_archivo_o_dir>
```

El resultado indica si el archivo fue formateado o ya se encontraba en su formato ideal.

Esta herramienta es integrable a VSCode. Para ello hay que descargarse el plugin. Luego en los settings necesitamos habilitar dos cosas:

- que el default formatter sea black
- que al guardar un archivo se aplique el format automaticamente

VSCode plugins

Los plugins de VSCode que recomendamos usar son:

Python

Incluye muchos plugins en uno: coloreo y autocompletado, estilo (pylance), python debugger y más

Black Formatter

Formatteo automático de código

Fin