Universidad Autónoma de Madrid

Departamento de Informática

Proyecto de Sistemas Informáticos
Práctica - 1

Equipo docente de PSI

Índice

1. Introducción y objetivos

El objetivo primordial de esta asignatura es desarrollar una aplicación web usando un framework de desarrollo profesional. Concretamente, trabajaremos con *Django*, un framework para el desarrollo de aplicaciones web en lenguaje *Python* que introduciremos en la Práctica 2. Por su parte, la Práctica 1 servirá de introducción al lenguaje *Python*, así como a *Git* (`https://git-scm.com/`), un software de control de versiones que se utilizará durante todo el curso. Como IDE, en la asignatura recomendamos el uso de *Pycharm* (`https://www.jetbrains.com/es-es/pycharm/`).

2. Descripción del trabajo a realizar

2.1. *Git*

Durante todo el curso se utilizará *Git* como herramienta de control de versiones, concretamente, la implementación de la plataforma *GitHub*. Para ello, lo primero que debéis hacer es conectaros a `https://github.com/` y registraros de forma gratuita en la plataforma. Vuestro nombre de usuario debe ser la inicial seguida de vuestro primer apellido siempre que sea posible. Si el usuario ya ha sido asignado añadir un número tras el apellido.

Como apéndice al enunciado de la práctica se muestra una descripción de los principales comandos de *Git*. Después de leer este apéndice, realizar el siguiente ejercicio como primera toma de contacto con *Git*.

EJERCICIO 1: Introducción a *Git*

1. Crear un repositorio llamado psi_numeroGrupo_numeroPareja_p1. El repositorio debe ser "Private".
2. Duplicar el repositorio localmente.
3. Modificar las restricciones de acceso para que los dos miembros de la pareja puedan modificar el repositorio.
4. Uno de los miembros de la pareja deberá subir el código del script `hello.py` disponible en *Moodle*.
5. En un ordenador diferente, el segundo miembro de la pareja (usando su usuario en el repositorio) deberá: (1) bajar el repositorio, (2) modificar el fichero para que escriba "Hola Mundo" en lugar de "Hello World" y (3) subir la modificación al repositorio.
6. El primer miembro de la pareja (usando su usuario en el repositorio) deberá: (1) actualizar el repositorio local, (2) crear un nuevo fichero llamado `hallo.py` idéntico al fichero `hello.py` pero cambiando el saludo "Hola Mundo" por "Hallo Welt" y (3) subir las modificaciones al repositorio.

2.2. *Python*

Python es un lenguaje de programación de propósito general que asumimos que la mayoría de vosotros desconoce. En esta práctica se proponen una serie de lecturas sencillas y ejercicios para familiarizarse con el lenguaje y disponer así de los conocimientos básicos para el resto de prácticas. En la práctica solo se pone el foco en ciertos aspectos básicos del lenguaje, pero *Python* ofrece muchas más posibilidades de lo que se indica aquí. En este sentido, se recomienda revisar y tener como referencia durante toda la asignatura la propia documentación del lenguaje `https://docs.python.org/3/index.html`. Aunque alguna de las lecturas y ejercicios propuestos están pensados para Python 2, durante el curso utilizaremos Python

3 (concretamente la versión 3.6.9, `https://docs.python.org/3.6/reference/`). Al nivel que vamos a utilizar *Python* en esta práctica, solo existen pequeñas diferencias que comentaremos más adelante en la descripción de los ejercicios.

EJERCICIO 2: Instalación y primera toma de contacto con *Python*

Leer las secciones *Python Set Up*, *Python Intro* y *Strings* del curso *Google's Python class* (`https://developers.google.com/edu/python/`). Tras realizar los ejemplos propuestos, implementar los ejercicios descritos en los ficheros `string1.py` y `string2.py` (están *Moodle*). Hacerlo de forma que vuestro código funcione en Python 3. Los únicos cambios entre versiones que os afectan son:

- `print` pasa de ser un comando a una función: `print 'hola'` → `print('hola')`.
- La división de enteros ha pasado a devolver una fracción en lugar de realizar la división entera: $5/2 = 2 \rightarrow 5/2 = 2{,}5$. La división entera se realiza ahora con el comando $5//2 = 2$.

EJERCICIO 3: Estructuras de datos

Leer las secciones *Lists*, *Sorting* y *Dicts and Files* del curso *Google's Python class* (`https://developers.google.com/edu/python/`). Tras realizar los ejemplos propuestos, implementar los ejercicios descritos en los ficheros: `list1.py` y `wordcount.py` (estos ficheros están disponibles en *Moodle*). Utiliza el fichero `alice.txt` como entrada del programa `wordcount.py`.

NOTA: Además de las lecturas simplificada del curso de *Google*, se recomienda la lectura del tutorial sobre estructuras de datos de la propia documentación de *Python* (`https://docs.python.org/3/tutorial/datastructures.html`).

EJERCICIO 4: Clases y objetos

Leer las secciones *Python Classes/Objects* y *Python Inheritance* del tutorial de *Python* de la *W3 schools* (`https://www.w3schools.com/python/python_classes.asp`). Implementar el ejercicio descrito en el fichero `vehicle.py` disponible en *Moodle*.

NOTA: Además de la lectura simplificada del curso de la *W3 schools*, se recomienda la lectura del tutorial sobre clases de la propia documentación de *Python* (`https://docs.python.org/3/tutorial/classes.html`).

EJERCICIO 5: Flake8

Probablemente estáis familiarizados con la aplicación `lint` que hace una comprobación sintáctica del código escrito en C. La utilidad `flake8` realiza la misma función en *Python*. Evalúa el código generado en los ejercicios anteriores con `flake8` y corrige el código siguiendo todas y cada una de las recomendaciones dadas por la aplicación.

NOTA: La solución a todos los ejercicios propuestos con su histórico de modificaciones debe subirse al repositorio *Git* creado en el ejercicio 1 dentro de un directorio de nombre `ejercicioX` (con X 2-4 según corresponda).

3. Material a entregar al finalizar la práctica

Esta práctica no requiere la entrega de ninguna memoria. Utilizando la aplicación *Moodle* se debe entregar un único fichero zip con la versión actualizada de tu respositorio *Git*, asegurándote de incluir la carpeta oculta `.git`. Incluir esta carpeta es requisito obligatorio para que la práctica se corrija.

4. Criterios de corrección

Puesto que la solución a los problemas a entregar está disponible en internet, la práctica se evaluará con la nota APTO o NO APTO. Para aprobar es necesario satisfacer todos y cada uno de los siguientes criterios:

- El resultado de TODOS los ejercicios propuestos debe ser correcto. El resultado es correcto cuando el mecanismo de autoevaluación incluido en cada ejercicio lo valide usando Python3.
- `flake8 <nombrefichero>` no devuelva ningún mensaje de error o advertencia.

NOTA 1: En esta práctica no se valorará la calidad del código producido siempre y cuando (1) produzca el resultado deseado, (2) implemente la funcionalidad solicitada y (3) no sea un reto para un ser humano con conocimientos de *Python* entender el código.

NOTA 2: El código usado para corregir esta práctica será el subido a *Moodle*. En ningún caso se usará el código existente en el repositorio de GitHub.

NOTA 3: Aquellos estudiantes que superen el test de conocimiento de *Python* tendrán automáticamente la calificación de APTO, estando exentos de realizar la entrega.

5. Apéndice: ¿Qué debes saber de *Git*?

Este apéndice muestra una breve descripción del funcionamiento de *Git*. En concreto, aquí no se va explicar la forma de trabajar con ramas, lo que puede ser un mecanismo interesante de conocer por el alumno. Para más detalles sobre el uso de *Git*, se recomienda revisar la documentación disponible en `https://git-scm.com/docs`.

Cuando trabajas con *Git*, lo haces sobre un repositorio en el que se almacena el histórico de versiones de tus ficheros. Existe un repositorio remoto alojado en el servidor de *Git*; y un repositorio local, que es una "copia" del repositorio remoto en un directorio local. Normalmente se empieza con un repositorio remoto clonado en la máquina de trabajo. Una vez descargado, todo el historial, todas las ramas y todas las versiones están disponibles sin conexión. Esto implica una separación entre guardar una versión de uno o varios archivos y subirla al servidor original para que otros puedan obtener esos cambios. La primera acción se llama *commit*, mientras que al acto de enviar esos datos al servidor se le llama *push*.

A continuación se muestran los principales comandos que se le pueden pasar a *Git* para interactuar e intercambiar información entre el repositorio local y el remoto:

- `git clone url_repositorio_remoto`
  Descarga y crea un enlace con el repositorio remoto en el directorio en el que nos encontramos.

- `git add <recurso>...`
  Modifica/crea uno o varios ficheros/directorios en el repositorio a partir del contenido del directorio de trabajo. Notar que *Git* obliga a hacer `add` tanto para nuevos elementos del repositorio como para archivos cambiados. Solo los elementos "añadidos" formarán parte del *commit*.

- `git rm <recurso>...`
  Elimina uno o varios recursos del repositorio y del directorio de trabajo.

- `git commit -m "Mensaje del commit"`
  Actualiza el repositorio local con todos los cambios realizados.

- `git status`
  Comprueba el estado del repositorio, mostrando tres listas de archivos: archivos modificados, archivos nuevos y archivos a incluir en el commit.

- `git checkout -- <recurso>`
  Sustituye un recurso en el directorio de trabajo por la versión en el repositorio.

- `git push`
  Sube los *commits* locales al repositorio remoto, de forma que las referencias remotas se actualizan con las referencias locales y se mandan al servidor los objetos necesarios para satisfacer estas referencias. Para tratar los posibles conflictos con los cambios realizados por otros usuarios del respositorio, lo recomendable es actualizar primero el respositorio local.

- `git pull`
  Actualiza el repositorio local con las últimas versiones del repositorio remoto. Al actualizar el repositorio pueden producirse conflictos porque los ficheros incluidos en la actualización han cambiado localmente. Dependiendo del tipo del conflicto es posible que haya editar y corregir el conflicto manualmente, dejando el archivo como debería estar después de aplicar todas modificaciones locales y remotas. Una vez resueltos todos los conflictos los ficheros en conflicto se deben volver a añadir (`git add`) y hacer *commit*.