Diseño de Aplicaciones para Internet (2018-2019)
Guión de Prácticas 1:
Python y Máquina Virtual

S. Alonso, J.M. Guirao
zerjioi@ugr.es, jmguirao@ugr.es

Resumen

Durante las prácticas de la asignatura *Diseño de Aplicaciones para Internet* (DAI) vamos a hacer uso intensivo de diversos lenguajes de programación. Entre otros, utilizaremos `Python`, un lenguaje de alto nivel sencillo, potente, libre, fácil de aprender, interpretado y multiplataforma (entre otras características). En esta primera práctica trataremos de resolver algunos problemas genéricos de programación usando dicho lenguaje para familiarizarnos con sus tipos de datos, estructuras de control, etc. Por otra parte haremos un primer uso de la máquina virtual para familiarizarnos con el control de máquinas "remotas".

Para aquellos que no tengan conocimientos sobre este lenguaje, existen numerosos manuales en Internet que permiten acercarse a la programación Python de manera sencilla y amena como por ejemplo: *"A Byte of Python"*: `http://swaroopch.com/notes/python/` o el manual de Google sobre Python: `https://developers.google.com/edu/python/`

1. Preparación del Entorno

Para las prácticas del curso, utilizaremos una máquina virtual con `Ubuntu 18.04`. De esta manera conseguimos:

- Aislar nuestro código del resto del ordenador, sin que interfiera lo que pueda instalarse antes o después.
- Conseguir un entorno **idéntico** para todos los participantes del curso.
- Tener nuestra aplicación lista para subirla a la nube en un IAAS (`https://azure.microsoft.com/es-es/overview/what-is-iaas/`)

Una opción fácil para manejar máquinas virtuales es utilizar `Vagrant` (`https://www.vagrantup.com/`), un software para facilitar la instalación, configuración y provisionamento de máquinas virtuales. Las siguientes instrucciones son para un sistema operativo anfitrión (`host`) `Ubuntu`, pero deben ser fácilmente adaptables a otros sistemas como `Windows` o incluso `MacOS`.

```
# Instalamos primero  virtualbox  y  vagrant  en el host:
  > sudo apt-get install virtualbox
  > sudo apt-get install vagrant
```

```
# Creamos el directorio para alojar la máquina virtual (MV):
  > mkdir DAI
  > cd DAI

# Creamos el archivo de configuración  Vagrantfile
# Desde este archivo podremos instalar interfaces, abrir puertos,
# crear carpetas compartidas, ajustar la memoria de la MV, instalar paquetes...
  DAI> vagrant init bento/ubuntu-18.04


# Ahora lanzamos la MV. Al ser la primera vez la bajará de Internet
# y luego la ejecutará
  DAI> vagrant up

# En algunas ocasiones en este punto parece colgarse la máquina vitual.
# Puede ser porque la configuración de red de la máquina virtual ponga
#que el "cable" de red está desconectado. Para solucionarlo, desde VirtualBox
# accedemos a la configuración de la máquina -> Red -> Adaptador 1 -> Avanzada y
# asegurarnos que "Cable Conectado" está señalado.

# Ya podemos conectarnos a la máquina virtual
  DAI> vagrant ssh

# Para salir de la máquina virtual (como en cualquier sesión SSH):
  vagrant> exit

# Dentro de la MV el directorio /vagrant se corresponde con el
# directorio DAI/ que creamos en nuestro host al principio. Por tanto podemos
# editar nuestro código dentro de dicha carpeta y tenerlo inmediatamente
# disponible en la MV.

# Es conveniente apagar la MV si no la vamos a usar:
  DAI> vagrant halt

# También podemos borrarla (normalmente no lo haremos salvo problemas graves)
  DAI> vagrant destroy
```

Una vez configurada nuestra máquina virtual entramos dentro:

```
DAI> vagrant ssh
```

Existen 2 versiones principales de `Python`: La `2.x` y la `3.x`. En la máquina virtual que estamos utilizando están instaladas las 2. Es conveniente saber que versión se ejecutará cuando llamemos a nuestro interprete de `Python`. Para eso podemos hacer:

```
> python --version
Python 2.7.15rc1
```

En caso de que la versión que nos aparezca sea la `2.x` es recomendable que cambiemos el intérprete para que ejecute la `3.x`:

```
> sudo ln -sfn /usr/bin/python3.6 /usr/bin/python
> python --version
Python 3.6.5
```

2. Problemas "Sencillos"

1. Programe un mini-juego de "adivinar" un número (entre 1 y 100) que el ordenador establezca al azar. El usuario puede ir introduciendo números y el ordenador le responderá con mensajes del estilo *"El número buscado el mayor / menor"*. El programa debe finalizar cuando el usuario adivine el número (con su correspondiente mensaje de felicitación) o bien cuando el usuario haya realizado 10 intentos incorrectos de adivinación.

2. Programe un par de funciones de ordenación de matrices (UNIDIMENSIONALES) de números distintas (burbuja, selección, inserción, mezcla, montículos...) (`http://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento`). Realice un programa que genere aleatoriamente matrices de números aleatorios y use dicho métodos para comparar el tiempo que tardan en ejecutarse.

3. La *Criba de Eratóstenes* (`http://es.wikipedia.org/wiki/Criba_de_Erat\%C3\%B3stenes`) es un sencillo algoritmo que permite encontrar todos los números primos menores de un número natural dado. Prográmelo.

4. Cree un programa que lea de un fichero de texto un número entero `n` y escriba en otro fichero de texto el `n`-ésimo número de la sucesión de Fibonacci (`http://es.wikipedia.org/wiki/Sucesi\%C3\%B3n_de_Fibonacci`).

5. Cree un programa que:

   - Genere aleatoriamente una cadena de [y].
   - Compruebe mediante una función si dicha secuencia está *balanceada*, es decir, que se componga de parejas de corchetes de apertura y cierre correctamente anidados. Por ejemplo:
     - [] → Correcto
     - [[][]] → Correcto
     - [][] → Correcto
     -][→ Incorrecto
     - [][[→ Incorrecto
     - []][[] → Incorrecto

3. Problemas "Complejos"

- Implemente el *Juego de la Vida* (`http://es.wikipedia.org/wiki/Juego_de_la_vida`). La salida de cada iteración debe guardarse en ficheros de texto con nombre consecutivo. Opcionalmente, se puede usar la biblioteca `graphics.py` para mostrar la evolución del juego como imágenes en vez de ficheros de texto. **Nota:** Si ejecutas los ejercicios en una máquina virtual

sin escritorio deberás hacer un X11 fordwarding (parámetro `-X` en el comando `ssh`) para poder visualizar ventanas con `graphics.py`. En Ubuntu es posible que necesite instalar el paquete `xorg` para tener un servidor `X11` y `python3-tk`:

```
> sudo apt-get install xorg python3-tk
> exit # Salir de la máquina virtual
> ssh vagrant@localhost -X -p 2222  # Entrar en la máquina virtual haciendo
                                    # X forwarding. User:pass vagrant:vagrant
```

- Realice un programa que cree un fichero de imagen que contenga una representación del *Conjunto de Mandelbrot* (`http://es.wikipedia.org/wiki/Conjunto_de_Mandelbrot`) entre unas coordenadas (x_1, y_1) y (x_2, y_2) que se le preguntarán al inicio del programa al usuario.
- Utilizando expresiones regulares (`http://docs.python.org/3.4/library/re.html`) realice funciones para:
  - Identifique cualquier palabra seguida de un espacio y una única letra mayúscula (por ejemplo: `Apellido N`).
  - Identifique correos electrónicos válidos (empiece por una expresión genérica y vaya refinándola todo lo posible).
  - Identifique números de tarjeta de crédito cuyos dígitos estén separados por `-` o espacios en blanco cada paquete de cuatro dígitos: `1234-5678-9012-3456` ó `1234 5678 9012 3456`.