ESCUELA POLITÉCNICA NACIONAL
FACULTAD DE INGENIERÍA DE SISTEMAS
ARQUITECTURA DE COMPUTADORES

Proyecto ICCD332 Arquitectura de Computadores

Profesor: Lenin G. Falconí M.Sc.

2024-08-07

Contents

1 City Weather APP

Este es el proyecto de fin de semestre en donde se pretende demostrar las destrezas obtenidas durante el transcurso de la asignatura de **Arquitectura de Computadores**.

1. Conocimientos de sistema operativo Linux
2. Conocimientos de Emacs/Jupyter
3. Configuración de Entorno para Data Science con Mamba/Anaconda
4. Literate Programming

1.1 Estructura del proyecto

Se recomienda que el proyecto se cree en el *home* del sistema operativo i.e. *home/<user>*. Allí se creará la carpeta *CityWeather*

```
pwd
```

El proyecto ha de tener los siguientes archivos y subdirectorios. Adaptar los nombres de los archivos según las ciudades específicas del grupo.

```
tree
```

Puede usar Emacs para la creación de la estructura de su proyecto usando comandos desde el bloque de shell. Recuerde ejecutar el bloque con `C-c C-c`. Para insertar un bloque nuevo utilice `C-c C-,` o `M-x org-insert-structure-template`. Seleccione la opción *s* para src y adapte el bloque según su código tenga un comandos de shell, código de Python o de Java. En este documento `.org` dispone de varios ejemplos funcionales para escribir y presentar el código.

```
echo 'Aquí va sus comandos'
```

```
Aquí va sus comandos
```

1.2 Formulación del Problema

Se desea realizar un registro climatológico de una ciudad $\mathcal{C}$. Para esto, escriba un script de Python/Java que permita obtener datos climatológicos desde el API de openweathermap. El API hace uso de los valores de latitud x y longitud y de la ciudad $\mathcal{C}$ para devolver los valores actuales a un tiempo t.

Los resultados obtenidos de la consulta al API se escriben en un archivo *clima-<ciudad>-hoy.csv*. Cada ejecución del script debe almacenar nuevos datos en el archivo. Utilice **crontab** y sus conocimientos de Linux y Programación para obtener datos del API de *openweathermap* con una periodicidad de 15 minutos mediante la ejecución de un archivo ejecutable denominado *get-weather.sh*. Obtenga al menos 50 datos. Verifique los resultados. Todas las operaciones se realizan en Linux o en el WSL. Las etapas del problema se subdividen en:

1. Conformar los grupos de 2 estudiantes y definir la ciudad objeto de estudio.
2. Crear su API gratuito en openweathermap

3. Escribir un script en Python/Java que realice la consulta al API y escriba los resultados en *clima-<ciudad>-hoy.csv*. El archivo ha de contener toda la información que se obtiene del API en columnas. Se debe observar que los datos sobre lluvia (rain) y nieve (snow) se dan a veces si existe el fenómeno.

4. Desarrollar un ejecutable *get-weather.sh* para ejecutar el programa Python/Java.[1]

5. Configurar Crontab para la adquisición de datos. Escriba el comando configurado. Respalde la ejecución de crontab en un archivo output.log

6. Realizar la presentación del Trabajo utilizando la generación del sitio web por medio de Emacs. Para esto es necesario crear la carpeta **weather-site** dentro del proyecto. Puede ajustar el *look and feel* según sus preferencias. El servidor a usar es el **simple-httpd** integrado en Emacs que debe ser instalado:

   - Usando comandos Emacs: `M-x package-install` presionamos enter (i.e. RET) y escribimos el nombre del paquete: simple-httpd
   - Configurando el archivo init.el

     ```
     (use-package simple-httpd
        :ensure t)
     ```

   Instrucciones de sobre la creación del sitio web se tiene en el vídeo de instrucciones y en el archivo Org-Website.org en el GitHub del curso

7. Su código debe estar respaldado en GitHub/BitBucket, la dirección será remitida en la contestación de la tarea

1.3 Descripción del código

En esta sección se debe detallar segmentos importantes del código desarrollado así como la **estrategia de solución** adoptada por el grupo para resolver el problema. Divida su código en unidades funcionales para facilitar su presentación y exposición.

Lectura del API

```
def adder(a,b):
    return a+b
print(adder(5,3))
```

Puede tener que borrar los dos puntos para que el resultado aparezca en el HTML. En mi caso no fue necesario. Pruebe.

```
8
```

Convertir *Json* a *Diccionario* de Python

```
print(adder(8,8))
```

Guardar el archivo csv

```
print(adder(8,-18))
```

[1] Recuerde que su máquina ha de disponer de un entorno de anaconda/mamba denominado iccd332 en el cual se dispone del interprete de Python

1.4 Script ejecutable sh

Se coloca el contenido del script ejecutable. Recuerde que se debe utilizar el entorno de **anaconda/mamba** denominado **iccd332** para la ejecución de Python; independientemente de que tenga una instalación nativa de Python

En el caso de los shell script se puede usar 'which sh' para conocer la ubicación del ejecutable

```
which sh
```

```
/usr/bin/sh
```

De igual manera se requiere localizar el entorno de mamba **iccd332** que será utilizado

```
which mamba
```

```
/home/leningfe/miniforge3/condabin/mamba
```

Con esto el archivo ejecutable a de tener (adapte el código según las condiciones de su máquina):

```
#!/usr/bin/sh
source /home/<user>/miniforge3/etc/profile.d/conda.sh
eval "$(conda shell.bash hook)"
conda activate iccd332
Python main.py
```

Finalmente convierta en ejecutable como se explicó en clases y laboratorio

```
#!/usr/bin/sh
Poner comando/s aquí
```

1.5 Configuración de Crontab

Se indica la configuración realizada en crontab para la adquisición de datos

```
*/t * * * * cd <City>Weather && ./get-weather.sh >> output.log 2>&1
```

- Recuerde remplazar <City> por el nombre de la ciudad que analice
- Recuerde ajustar el tiempo para potenciar tomar datos nuevos
- Recuerde que `2>&1` permite guardar en `output.log` tanto la salida del programa como los errores en la ejecución.

2 Presentación de resultados

Para la pressentación de resultados se utilizan las librerías de Python:

- matplotlib
- pandas

Alternativamente como pudo estudiar en el Jupyter Notebook CityTemperatureAnalysis.ipynb, existen librerías alternativas que se pueden utilizar para presentar los resultados gráficos. En ambos casos, para que funcione los siguientes bloques de código, es necesario que realice la instalación de los paquetes usando `mamba install <nombre-paquete>`

2.1 Muestra Aleatoria de datos

Presentar una muestra de 10 valores aleatorios de los datos obtenidos.

```
import os
import pandas as pd
# lectura del archivo csv obtenido
df = pd.read_csv('/home/leningfe/PythonProjects/QuitoWeather/clima-quito-hoy-etl.csv')
# se imprime la estructura del dataframe en forma de filas x columnas
print(df.shape)
```

Figure 1: Lectura de archivo csv

Resultado del número de filas y columnas leídos del archivo csv

```
(57, 30)
```

```
table1 = df.sample(10)
table = [list(table1)]+[None]+table1.values.tolist()
```

Figure 2: Despliegue de datos aleatorios

dt	$\text{coord}_{\text{lon}}$	$\text{coord}_{\text{lat}}$	$\text{weather}_{\text{0description}}$	...	$\text{main}_{\text{temp}}$	name	cod
2024-08-03 21:57:57	-78.5249	-0.2299	overcast clouds	...	8.53	Quito	200
2024-08-04 10:26:16	-78.525	-0.2299	overcast clouds	...	16.53	Quito	200
2024-08-04 09:15:02	-78.5249	-0.2299	overcast clouds		14.53	Quito	200
2024-08-06 10:05:50	-78.5211	-0.2309	few clouds		14.66	Quito	200
2024-08-03 02:43:26	-78.5249	-0.2299	scattered clouds		7.53	Quito	200
2024-08-04 22:50:26	-78.5249	-0.2299	scattered clouds		9.53	Quito	200
2024-08-03 12:52:29	-78.5211	-0.2309	few clouds		20.66	Quito	200
2024-08-03 10:54:26	-78.5211	-0.2309	clear sky		15.66	Quito	200
2024-08-02 23:51:42	-78.5211	-0.2309	broken clouds		8.66	Quito	200
2024-08-03 02:13:58	-78.5249	-0.2299	scattered clouds		7.53	Quito	200

2.2 Gráfica Temperatura vs Tiempo

Realizar una gráfica de la Temperatura en el tiempo.

El siguiente cógido permite hacer la gráfica de la temperatura vs tiempo para Org 9.7+. Para saber que versión dispone puede ejecutar `M-x org-version`

```
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
# Define el tamaño de la figura de salida
fig = plt.figure(figsize=(8,6))
plt.plot(df['dt'], df['main_temp']) # dibuja las variables dt y temperatura
# ajuste para presentacion de fechas en la imagen
plt.gca().xaxis.set_major_locator(mdates.DayLocator(interval=2))
# plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d'))
plt.grid()
# Titulo que obtiene el nombre de la ciudad del DataFrame
plt.title(f'Main Temp vs Time in {next(iter(set(df.name)))}')
plt.xticks(rotation=40) # rotación de las etiquetas 40°
fig.tight_layout()
fname = './images/temperature.png'
plt.savefig(fname)
fname
```

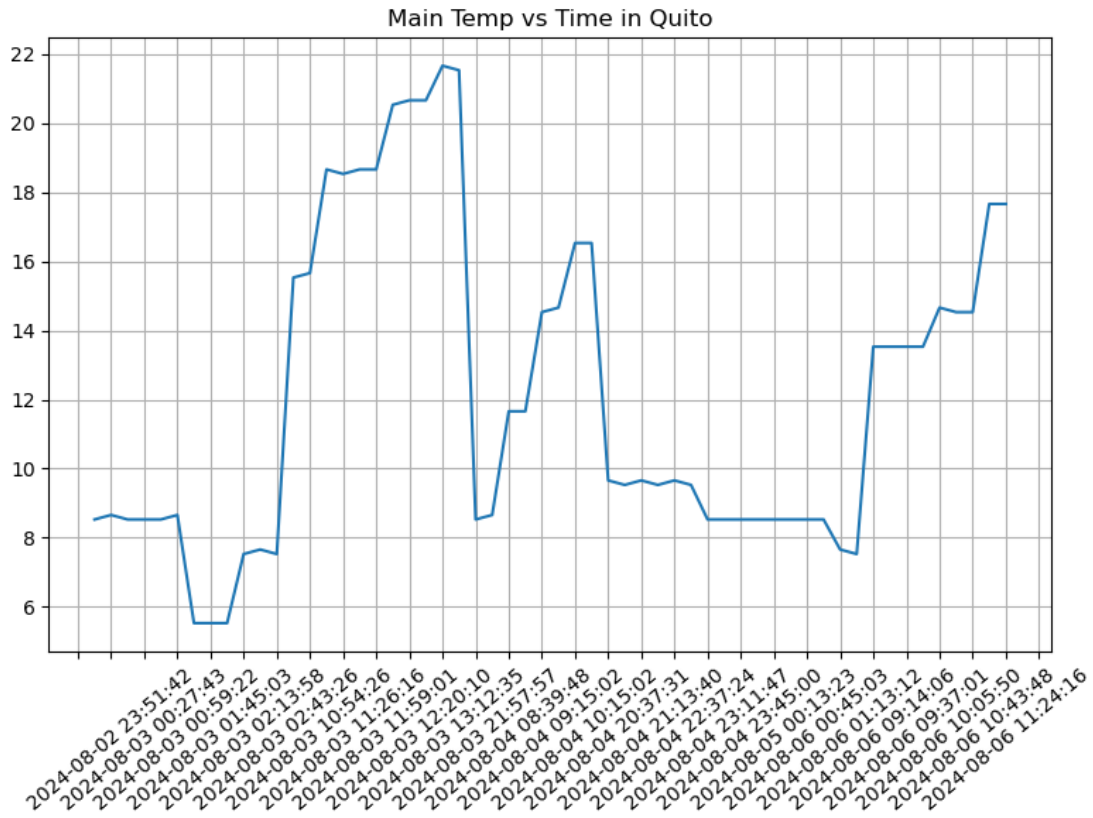

Figure 3: Gráfica Temperatura vs Tiempo

Debido a que el archivo index.org se abre dentro de la carpeta *content*, y en cambio el servidor http de emacs se ejecuta desde la carpeta *public* es necesario copiar el archivo a la ubicación equivalente en `/public/images`

```
cp -rfv ./images/* /home/leningfe/PythonProjects/QuitoWeather/weather-site/public/images
```

2.3 Realice una gráfica de Humedad con respecto al tiempo

2.4 Opcional Presente alguna gráfica de interés.

3 Referencias

- presentar dataframe como tabla en emacs org
- Python Source Code Blocks in Org Mode
- Systems Crafters Construir tu sitio web con Modo Emacs Org
- Vídeo Youtube Build Your Website with Org Mode