Infrastructure as Code-Walkthrough

Python Simple Logging

Contents

Last Updated: 13 October 2022

A **walkthrough** is intended to bring you through a technical exercise. A walkthrough shows you how I completed a task in a particular context, on specific systems, at a point in time. The document is a good basic guide, but you should always confirm that the information is current and if a more recent best practice exists, you should make yourself aware of it.

Introduction

This walkthrough is provided to demonstrate how to get started with logging in Python. I am going to use simple techniques, rather than the logging libraries. And I need to cover time and file access to make this work.

Deliverables

You will be required to keep a list of commands with notes as to why you are using them as well as results of any testing you have done. You may be asked to present these notes as evidence of work completed or as part of a lab book. Check with your lecturer!

Prerequisites

1. I will carry out my exercises using a Windows 11 desktop. You can carry out this work on almost any modern platform.
2. You should have already installed Python as per my notes.
3. You should have already installed VSC as per my notes.
4. You have created a directory to keep example files in and you are ready to code.
   - **a.** This could be on your OneDrive, in these examples, I am using **OneDrive\Python\Exercises_10**

Time in Python

I am creating a first python file called **mytime.py** for this exercise.

There are several standard libraries we can use in Python to handle time. The module datetime [2] is one of the most used.

I am going to let you read the detail, rather than explaining everything here.

```
from datetime import datetime as dt
# Get the current time, returns a value like 2022-10-09 17:46:45.151666
today = dt.now()
print(today)
# Get Unix time, returns a value like 1665566809.057217
unix_epoch_time = dt.timestamp(today)
print(unix_epoch_time)
```

Once you have read the detail, you should be able to calculate things like timedelta, the difference between two times. The strftime() method is handy for making useful strings from the datetime object. Rather than using a function to return the weekday, I can use

weekday = dt.now().strftime("%A")

Similarly, to get a month, I can use

month = dt.now().strftime("%B")

Hours, minutes, etc. all have their own codes available.

Exercise

Add to the code above and print human readable values for current time (hours, minutes, seconds) and date (year, month, day).

Logfiles

I need to detect my OS before I start my logfiles program. The following code from Walkthrough 6 is saved as **EXERCISES_10/os_utilities.py**

```
""""
OS utilities, forked from the Comm module of SD-Node, written c. 2017
Tested with Python >=3.6
By: JOR
  v0.1   26AUG21
"""
import platform

def detect_os()->str:
  # Detect the OS in use
  return platform.system()

if (__name__ == '__main__'):
  print(f"This module is called {__name__} and executes as a standalone script")

  # Check the OS in use, lower case
  my_os = detect_os()
  my_os = my_os.lower()

  # Parse the response
  if my_os == "windows":
    print("Your system is Windows")
  elif my_os == "linux":
    print("Your system is Linux")
  elif my_os == "darwin":
    print("Your Apple system is MacOS")
  elif my_os == "cygwin":
    print("Your Apple system is MacOS")
  elif my_os == "aix":
    print("Your IBM system is AIX")
  else:
    print(f"Unidentified system = {my_os}")
else:
  pass
  #print(f"This module is called {__name__} and is being called by another script")
```

Note that at the end, I have commented out my diagnostic text and added a **pass** statement.

Interestingly, I can copy and paste from the **if (__name__ == '__main__'):** section if I want to use functions in another program.

When I'm saving logfiles, I like to have unique filenames. I wrote the following file long ago in my utilities package and I am copying it now as **EXERCISES_10/file_utilities.py**

```
"""
File utilities, forked from the Comm module of SD-Node, written c. 2017
Tested with Python >=3.6
By: JOR
  v0.1    26AUG21
"""

from datetime import datetime as dt
import sys, csv

def path_name():
  # Operating system dependent stuff
  this_os = sys.platform
  if this_os == 'win32':
    return './logfiles/'
  elif this_os == 'linux':
    return '/home/pi/logfiles/'
  else:
    print(f'Unsupported OS: {this_os}')
    exit(0)

def log_file_name(extension):
  """
  Create a file name in the logfiles directory, based on current data and time
  Requires the computer to have an RTC or synched clock
  """
  now = dt.now()
  # Linux
  file_name = '%0.4d%0.2d%0.2d-%0.2d%0.2d%0.2d' % (now.year, now.month, now.day, now.hour, now.minute, 
now.second)
  return file_name + extension

if (__name__ == '__main__'):
  print(f"This module is called {__name__} and executes as a standalone script")
  log_path = path_name()
  filename = log_file_name(".log")
  print(log_path + filename )
else:
  pass
  #print(f"This module is called {__name__} and is being called by another script")
```

The path that I use to store my log files is dependent on the operating system I'm using. As you can guess from the Linux path, this is intended for a Raspberry Pi.

Once again, I can copy and paste from the **if (__name__ == '__main__'):** section if I want to use these functions in another program.

I can bring all this together now in a main program.

```
from file_utilities import path_name, log_file_name
from os_utilities import detect_os


# Check the OS in use, and figure out a log file name and path
this_os = detect_os()
log_path = path_name()
filename = log_file_name(".log")
print(log_path + filename )
```

On my Windows machine, this returns

./logfiles/20221012-142149.log

In three lines of code, I reliably get a logfile name and path in any supported OS.

Interestingly, if I sort by filename, the logfiles will sort in chronological order.

I can use the above code for any project where I need to log data. This is my own standard for logging instrumentation data, you can find this code in many places in the real world!

Now I need to do something useful, in this example I’m going to log CPU information.

CPU Information

I need to do **pip install psutil** in the terminal to get the library I need. Do some reading about this library at https://pypi.org/project/psutil/

In **os_utilities** I add the line **import psutil** beside the other import statements.

Then I add a function **cpu_load()** to gather useful information, it almost doesn't matter what I gather, this is meant as an example only.

```
def cpu_load():
    # Return significant numbers relating to the CPU
    #print(f"Number of CPUs: {psutil.cpu_count()}" )
    #print(f"CPU load: {psutil.cpu_percent()}")
    return(psutil.cpu_count(), psutil.cpu_percent())
```

I used the print statements for testing and then commented them out.

The main program

Back in Walkthrough 7 we learned how to do basic file handling, I'm going to reuse that code now.
I modify my main program and call it **cpu_log.py** and I change the file extension to **".csv"**.

I use **time.sleep** to create a one second delay.

If you have a program running in an endless loop, in VSC, click into the terminal window and press **[ctrl][c]** to exit. If you run code in the Linux or Windows terminal window, the same approach works.

```
"""
File utilities, forked from the Comm module of SD-Node, written c. 2017

Tested with Python >=3.6

By: JOR
    v0.1    26AUG21

""""
from file_utilities import path_name, log_file_name
from os_utilities import detect_os, cpu_load
from time import sleep

# Check the OS in use, and figure out a log file name and path
this_os = detect_os()
log_path = path_name()
filename = log_file_name(".csv")

# Loop forever
while True:
    # Sleep for 1 second
    sleep(1)
    # Get a time stamp for this line
    timestamp = log_file_name("")
    # Get some information
    information = cpu_load()
    # Create a line for the logfile, convert the integer values to string
    logline = timestamp + ":" + str(information[0]) + "," + str(information[1]) + "\n"
    # Now write it to the logfile
    try:
        with open(filename, "a") as file_handle:
            print(f"logging to {filename}")
            file_handle.write(logline)
    except IOError as err:
        print(f"IOError was {err}")
    except EOFError as err:
        print(f"End of file error was {err}")
    except OSError:
        print("OS Error")
    except:
        print("General Error")
```

Note that I used the **log_file_name()** function with no extension to generate a timestamp for each line in the logfile.

Read and understand this program, then run it.

Testing

Run the code for a few minutes to verify it saves data once per second.

Do that more than once, verify that a new logfile is created each time.

A comma space delimited file is a typical way to exchange sensor data. You should be able to open and process the log file in Excel, check that now.

Finally

There are much more elegant ways to do some of this.

For systems logging, read [2].

There is a specific library for writing CSV files [3].

I wanted to do all this the hard way!

I have code in navigation systems, survey equipment, satellite tracking systems, instrumentation systems, etc. that has used similar, very simple scripts to that which I showed here.

I have a rule with systems which run 24x7 remotely……simplicity!