ArcGIS automation through scripting with Python

3_Packages

Step 1: The *os* Package

The os (operating system) module allows you to interact with the main operating systems, be it a Windows machine, Mac or Linux. Largely, os is system independent and provides ways that you can code a generic interface to many functions.

First, let's import os:

```
# The os package gives you functions that allow you to interface with the operating
system:
import os
```

The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux. For example:

```
# Part b - Other interesting functions you can use within Python
path = r"test_dir"

# Create a directory
os.mkdir(path)

# Return a list of the entries in the directory given by path.
list = os.listdir(path)
print list

# Rename the file or directory src to dst.
os.rename("test_dir", "test_dir2")

# Remove directories recursively.
os.removedirs(path + "2")

# # Remove (delete) the file path.
os.remove(path)

# # Remove (delete) the directory path.
os.rmdir(path)
```

TASKS

```
# Task - Use os.system to make a directory in your current directory, check if it is
created delete it and check again.
```

```
# Bonus Task 2 - Use os to make a directory in your root directory, add a subdirectory
inside it, check if it is created, delete the subdirectory and the main directory.
Check if the main directory exists, print "namrdir EXISTS" or "namrdir NOT EXISTS"
```

Step 2 – The *sys* Package

The system package gives you functions that allow you to interface with Python irrespective of the platform we are using.

```
import sys
```

```
# For example, we can find out the version of the software:
print sys.version

# Locate our Python Executable path
print sys.executable

# Read and write to the interpreter directly (similar to print function, but a bit
more powerful)
sys.stderr.write('This is stderr text\n')
sys.stderr.flush()
sys.stdout.write('This is stdout text\n')
```

But perhaps the most important function is sys.argv, which will allow you to add commands to the interpreter from an external file. I use this when executing my Python code from a *.bat file, as you can write:

C:\Python27\ArcGIS10.5\python.exe Step_2.py ARGUMENT1 ARUGMENT2

Try executing the Step_2.bat file that I have provided, it contains a simple command:

```
C:\Python27\ArcGIS10.5\python.exe Step_2.py IamArgument
pause
```

By calling this, and using:

```
sys.argv[1]
```

We can capture IamArgument and use this argument in our script. Ideal for adding variables to the script without editing the main code. In the IamArgument example, all we are doing is capturing and printing the argument:

```
def main(arg):
    print("My argument: " + str(arg))
main(sys.argv[1])
```

TASKS

```
# Task - Using sys.argv write a small code block to read in 3 arguments to your Python
file store each one in a list and iterate through them, printing each with the text
"Argument 1 = ", "Argument 2 = ". Hint, you will need to edit the *.bat file to
include the arguments.
```

```
# Bonus Task 2 - Code your Scrabble score work from Coding Challenge 2 to use 3
sys.argv inputs
```

Step 3 – Creating and writing files

Creating and writing to text files could not be any more simple in Python, for example:

```
# Creating a text file
file = open("testfile.txt", "w")

file.write("Hello World\n")
file.write("This is a line in our text file\n")
file.write("and this is another line.\n")
file.close()
```

See we use file = open(… to open our file, the “w” indicates *write*. We then write to the file the various strings we require. Note adding \n (newline), if we don’t it writes all on one line.

If we create this file again, using different words, you will notice it overwrites:

```
file = open("testfile.txt", "w")

file.write("Goodbye World\n")
file.write("This is not a line in our text file\n")
file.write("and this is not another line.\n")
file.close()
```

You need to be careful with this, as once you close/reopen the file, it will create it as a new one. To add to a file, the change is subtle.

```
file = open("testfile.txt", "a")
file.write("Wooahhh, I am a line added later\n")
file.close()
```

Reading files is easy, but again a subtle change:

```
file = open("testfile.txt", "r")
file_contents = file.read()
print (file_contents)
file.close()

# You can also read a line:
file = open("testfile.txt", "r")
file_contents = file.readlines()
counter = 1
for i in file_contents:
    print str(counter) + ": " + i.rstrip() #.rstrip() removes new line characters i.e.
\n
    counter = counter + 1
file.close()
```

You can also read a line, and iterate through them, note that i.rstrip() removes new line characters i.e. \n, allowing me to print without a lot of extra lines appearing on the screen.

```
file = open("testfile.txt", "r")
file_contents = file.readlines()
counter = 1
for i in file_contents:
    print str(counter) + ": " + i.rstrip()
    counter = counter + 1
file.close()
```

TASK

```
# Task - Using the provided text file Task_Step_3.txt, within the Tasks directory,
iterate through the existing file, and find which line numbers the following words are
on: Limpet, Mongoose, Spider monkey, Zebra. Hint, you will likely fail to match the
whole word, as a new line is appended in the file, so as you iterate through each line
you should match line.rstrip(), as this removes the \n
```

Step 4 – The *csv* Package

One of the more ubiquitous open data file formats is the CSV file, aka comma separated values. They are best because they don't use any proprietary formats, e.g. "Excel", so you can be assured that this format will be readable many years from today, even if the software you used to create it is list.

I am assuming that everyone is used to a CSV format, e.g.:

```
### column 1 name,column 2 name, column 3 name
### first row data 1,first row data 2,first row data 3
### second row data 1,second row data 2,second row data 3
```

First import csv:

```
import csv
```

Using the Step_4.csv file I have provided, which is a list of countries and their populations through several years, we can process this file using the csv package. In this example, I am just using a counter to add up the total number of lines, once done, we print it. I also threw in a bonus of asking if we were at the first line, if so, give me the column names.

```
with open("Step_4.csv") as population_csv:
    csv_reader = csv.reader(population_csv, delimiter=',')

    line_count = 0

    for row in csv_reader:
        if line_count == 0:
            print "Column names are: " + str(row)
            line_count += 1
        line_count += 1

print("Processed " + str(line_count) + " lines.")
```

We can also do some basic tasks, for example, let's sum the "Population" column:

```
with open("Step_4.csv") as population_csv:
    headerline = population_csv.next()
    total = 0
    for row in csv.reader(population_csv):
        total += float(row[3]) #3 = 4th column, in this case, population
    print format(total, 'f') # format prints as float
    print total  # without we print as engineering notation
    # Now look at the data, 3 hundred billion? Ah, we have multiple years..
```

TASK

```
# Task - Using Step_4.csv, complete the following, 1) Make a list of the years and the
countries in the dataset, how many years and countries are there? 2) Calculate the
global population for every year in the dataset. 3) Split the global dataset into a
file for each country, with the filename the name of the country, in a new directory
using only Python
```