Unit 9. Modules and Files

Héctor D. Menéndez

Endless Science

endsci.net

Index

Index

Modules or Libraries

In our previous class, we were talking about functions. Normally functions develop a specific task but they **tend to be related** to other functions.

For example, some functions related to files might be `read`, `write`, `seek`, `open` or `close`.

Organising software requires to put all these functions together in the same **module**.

Modules or Libraries

A module or library is an abstraction of multiple functions performing tasks of the same nature.

For example, we can have the library `files` and inside of this library we would have the functions: `read`, `write`, `seek`, `open` or `close`.

In Python, when we need a library, we have to **import it**. There are two options: you can either import the whole library or just a specific function.

Import Example

Imagine that you want to import mathematical functions or constants (`sqrt` or π), you can use `import math`. Then, you will be able to use these elements from the library.

```
1 import math
2
3 print(math.pi)
4 print(math.sqrt(2))
```

Listing 1: Math Imports

Import Example

You can also improve the import if you do not want to use the library name all the time. For that you specify that, from the library `math`, you want to import `pi` and `sqrt`. Then, you will be able to use these elements without the library's name.

```
1 from math import pi,sqrt
2
3 print(pi)
4 print(sqrt(2))
```

Listing 2: Better Imports

Import Example

If for any reason, you want to import every mathematical function and constant from the library, you can use $*$ instead of the functions' names, but this is not recommended.

```
1 from math import *
2
3 print(pi)
4 print(sqrt(2))
```

Listing 3: All Imports

Import Example

You can also create an alias for your library to simplify its name, for example, we will call `m` to the `math` library.

```
1 import math as m
2
3 print(m.pi)
4 print(m.sqrt(2))
```

Listing 4: Alias Imports

Index

Basic Modules

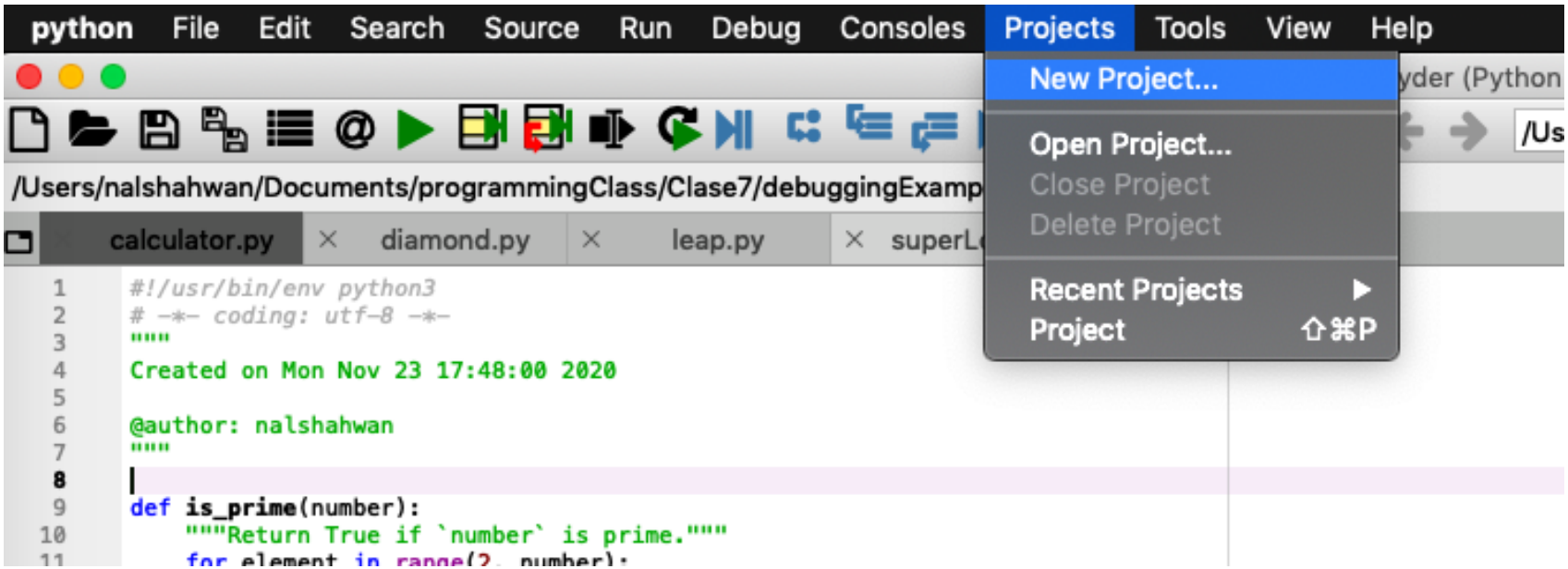
In the same way that you can use external libraries, you can define your own.

For that, you just need to put the functions that you want to group into a Python file and import it in a similar way.

As these modules are going to be basic and local, they need to be in the **same folder** of your main module.

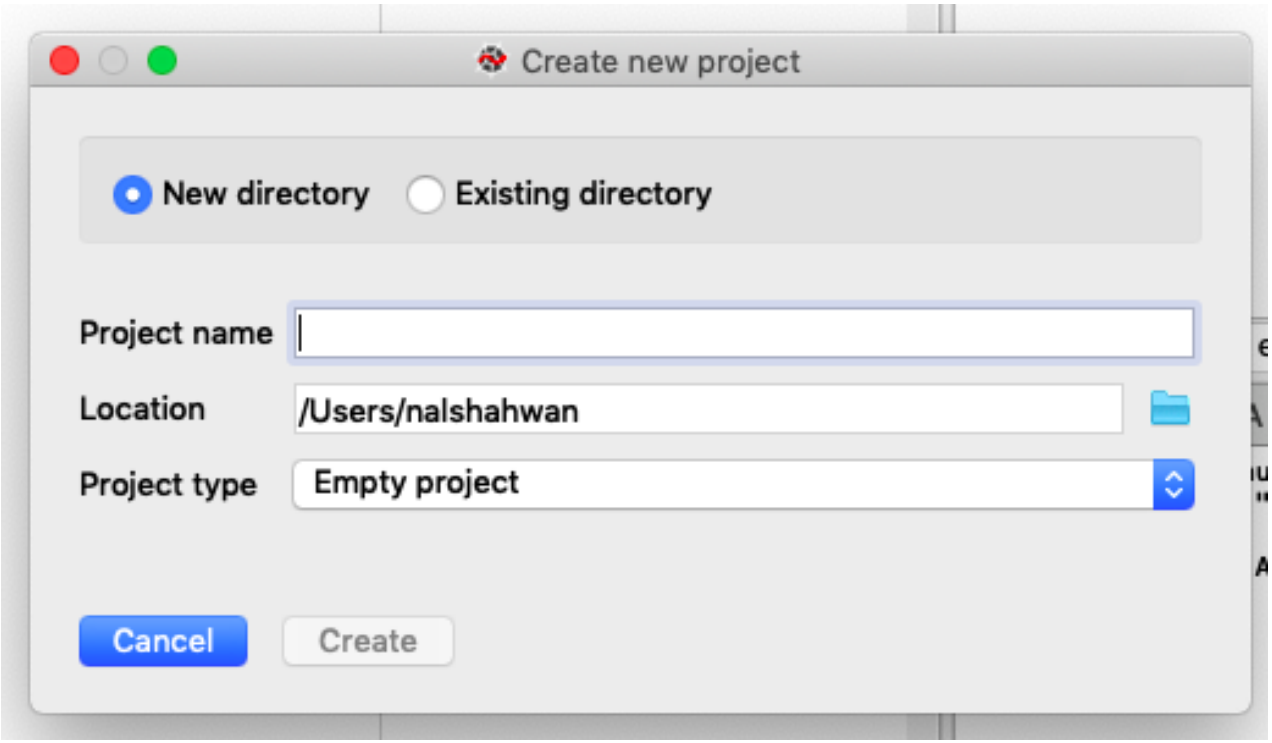
Our First Modules

Spyder helps us to organize modules. It creates an abstraction called project. Using the Spyder's interface, we are going to create a new project.



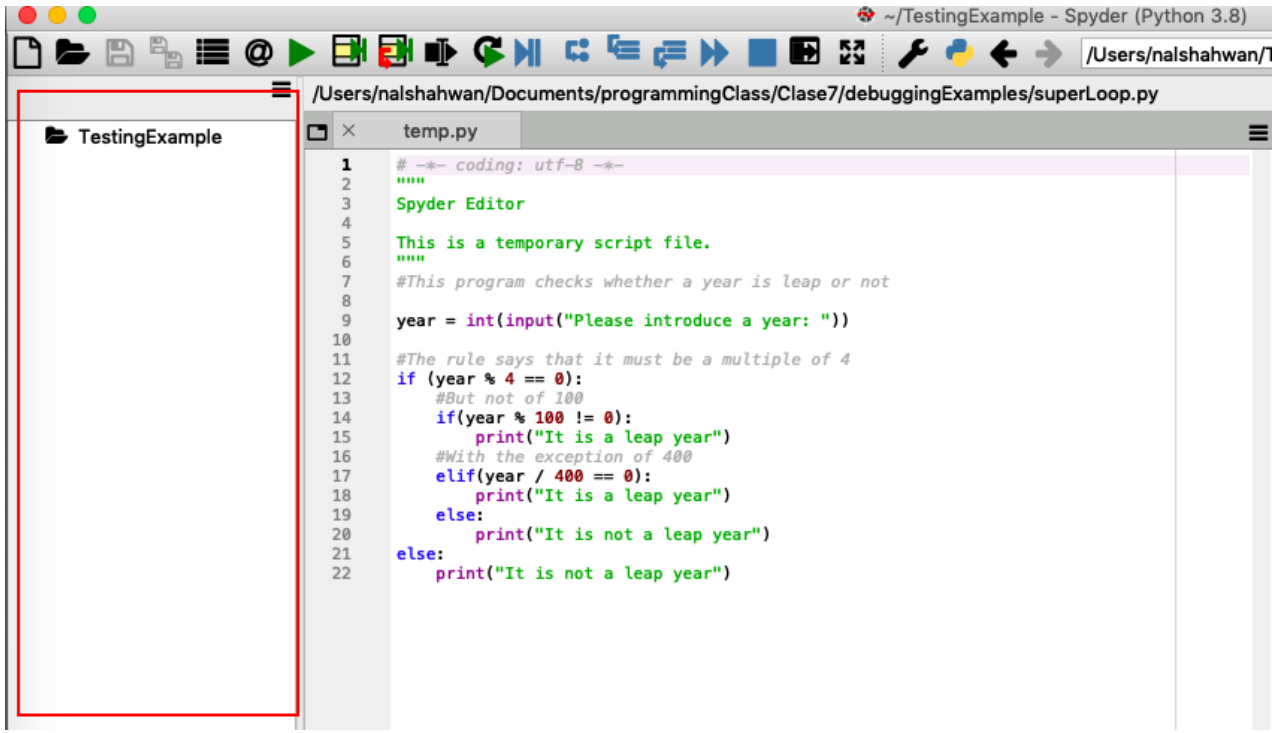
Our First Modules

Now, you can name the project and choose its location. Remember that every file must be inside of the project's folder. If you already have a previous project, use "Existing Directory"



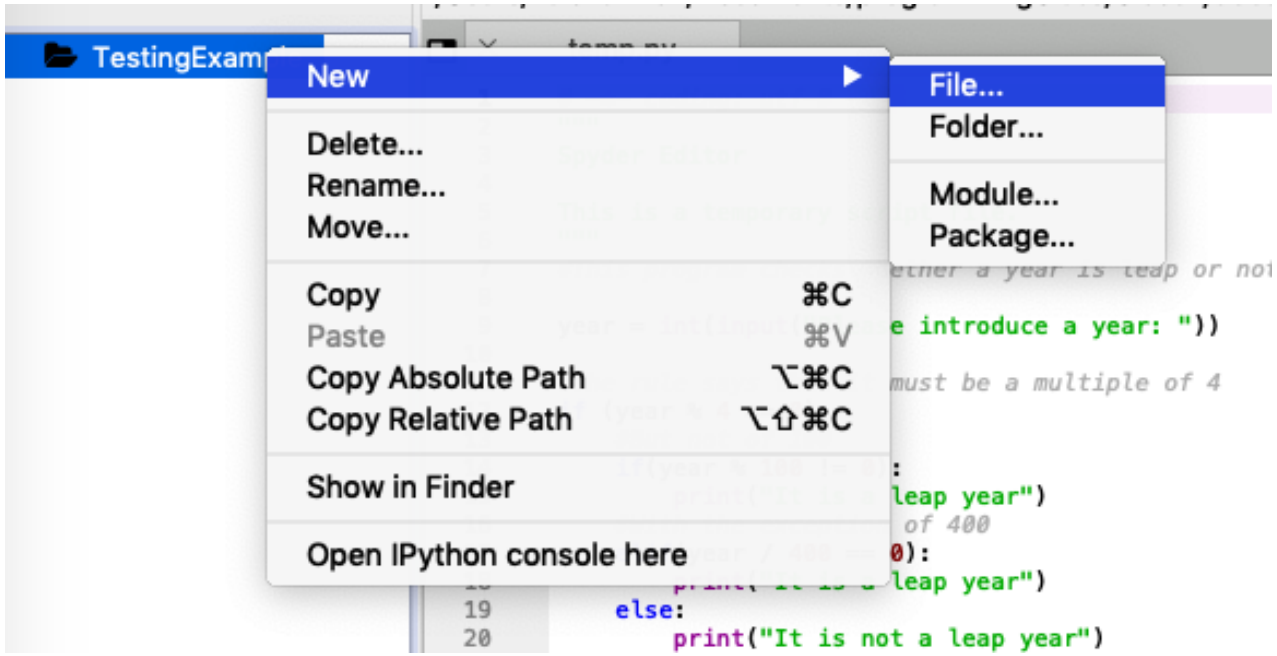
Our First Modules

Once you create a project, you will have a new section in the left hand side where your project files will be. Every file is a Python module.



Our First Modules

Now, you can create a new files for each library you want to add. Remember to put the files inside of the project's folder.



A Module of Prime Numbers

We are going to create a module that will work with prime numbers. Create a new file called `primes.py`. It will contain two functions: `is_prime` and `next_prime`. First, we will create the `is_prime` function.

```
1 def is_prime(number):
2     for i in range(2,number//2+1):
3         if(number % i == 0):
4             return False
5     return True
```

Listing 5: is_prime function

A Module of Prime Numbers

Now, create a new module or file called `main.py` (the name is not relevant). Inside of this module import the primes module and call it.

```
1  import primes
2
3  if(primes.is_prime(5) == True):
4    print(5," is a prime number")
5  else:
6    print(5," is not a prime number")
7  if(primes.is_prime(4) == True):
8    print(4," is a prime number")
9  else:
10   print(4," is not a prime number")
```

Listing 6: is_prime main

A Module of Prime Numbers

Now, create the second function in `primes.py`. This function will calculate the next prime.

```
1 def next_prime(number):
2     while (not is_prime(number+1)):
3         number=number+1
4     return number+1
```

Listing 7: next_prime function

A Module of Prime Numbers

You can also use it in `main` or change the way you import it.

```
1 from primes import next_prime
2 import primes
3
4 print("The next prime of 4 is: ", primes.next_prime(4))
5 print("The next prime of 5 is: ", next_prime(5))
```

Listing 8: next prime main

Index

Files

One of the most relevant concepts in computers and programming is files. If you remember your Linux classes, you must know that **in Linux everything is a file**.

Files are normally used to represent **data storage** inside of systems, but they can also be used as **communication channels**, for example via the network.

The most common operations related to files are `read`, `write`, `seek`, `open` and `close`.

Using Files in Python

Create a new project with Spyder, and inside of that project, create a text file called: "*myfile.txt*".

Put any content inside of the file, but make sure it has around 10 to 20 lines.

Now, we are going to read the file.

Reading Files in Python

First, we open the file with the read ("r") option. Now, we can read the complete file ("`read`") or just parts of it ("`readline`"). At the end, we need to close the file.

```
1 f = open("myfile.txt", "r")
2 print(f.read()) #Reads all file
3 print(f.readline(5)) #Reads the 5th line
4 f.close() #closes the file
5
6 f = open("myfile.txt", "r")
7 print(f.readline()) #Reads the next line
8 print(f.readline()) #Reads the next line
9 f.close() #closes the file
```

Listing 9: read a file

Writing Files in Python

We can also write a file in order to save some relevant information from our programs, for example, we will create a file called "*mylist.txt*".

When we open a file for writing, Python **creates** the file if the file does not exist. If it exists, it will be **overwritten**, and its original content will **disappear**.
We need to use the option "w", when we open the file.

Writing Files in Python

First, we open the file with the write ("w") option. Now, we can write inside of the file ("`write`"). At the end, we just close it.

```
1 f = open("mylist.txt", "w")
2 for number in range(0,10):
3   f.write(str(number)+"\n")
4 f.close() #closes the file
5
6 f = open("mylist.txt", "w")
7 for number in range(0,100):
8   f.write(str(number)+"\n")
9 f.close() #closes the file
```

Listing 10: write a file

Other File Options

It is possible to use different file options in Python and combine them. Some examples are:

Option	Meaning
r	Only reads a file
rb	Only reads a file in binary format
w	Only writes a file
r+	Reads and writes a file. Set the pointer at the beginning.
w+	Writes and reads a file, if it exists it overwrites the file.
a	Writes a file, if it exists it appends information at the end of the file.
a+	Reads and writes a file, if it exists it appends information to the file.

Seeking into a File

We can also move inside of a file to seek for relevant information.

There are three main positions: `SEEK_SET` (beginnig), `SEEK_CUR` (current) and `SEEK_END` (end). We need to import the os library to use them, because their encoding depends on the **operating system**.

Normally, this is used for binary files and we set a number of bytes before or after these specific points.

Seeking into Python Files

```
1  import os
2
3  f = open("mylist.txt", "rb")#Uses the file as a binary
4  f.seek(0, os.SEEK_SET) #Beginning of the file
5  print(f.readline())
6  f.seek(0, os.SEEK_END) #End of the file
7  print(f.readline())
8  f.seek(4, os.SEEK_SET) #Four characters after the
       beginning of the file
9  print(f.readline())
10 f.seek(10, os.SEEK_SET) #Four characters after the
       beginning of the file
11 print(f.readline())
12 f.seek(-4, os.SEEK_CUR) #Four characters before the
       current position
13 print(f.readline())
```

Listing 11: seeks into a file

Index

Exercise 1: My Sort Module

Create a module called mysort.py with three functions:

- `is_sorted`: checks whether a list of numbers is sorted and returns True or False.
- `swap`: takes a list a two positions and exchanges the elements on those positions.
- `bubbleSort`: sorts a list using the bubble sort algorithm. Test the function with an unsorted list. It returns the sorted list.

```
1 import mysort
2
3 myList=[3,1,5,2]
4 if(mysort.is_sorted(myList)):
5   print("The list is sorted")
6 else:
7   print("The sorted list is:",mysort.bubbleSort(myList
    ))).
```

Exercise 2: Writing Files

Write a file with a list of 100 random numbers in each line. Use the function randint from the module random, and set the generation boundaries for the random numbers between 0 and 1000.

Exercise 3: Reading Files

Read the previous file, store the list of numbers into a list and sort them with your version of mysort module.