Unit 11. Debugging and Testing

Héctor D. Menéndez

Endless Science

endsci.net

Index

Index

Debugging

Debugging executes the program in a step-by-step basics, showing information about the instructions and current program state.

It is normally used to verify the logic of the program's execution and to understand the nature of bugs.

The debugging process normally requires human intervention to understand the program's execution.

Debugging: Breakpoints

The process of debugging in 6 main operations and the concept of **breakpoint**.

If a program is running in debugging mode, a breakpoint will stop its execution and give control to the analyst to check the program's state.

Debugging: Operations

Once the control of the program passes to the analyst, she can perform 6 operations on the code.

Operation	Description
Run	Runs the program from the beginning.
Step	Runs the next step of the program, if their is a function call, it goes on inside of the function's code.
Next	Runs the next step of the program, if their is a function call, it skips the function's code.
Break	Sets a breakpoint in a specific line.
Continue	Continues with the normal program execution.
Stop	Finishes the execution.

Debugging: Process

The process of debugging normally starts with an error that we want to reproduce, normally produced by a specific input.

Based on the error, we set a breakpoint in a specific line where we want to analyse the program's state.

From that line, we perform the execution step by step until we have an idea about which part of the code is wrong.

Once we finish, we continue to identify the next bug.

Debugging: Example

In this case, we have a program with a bug and we aim to identify. We suspect that is related to the final comparison so we set the breakpoint at line 4.

```
1 def is_prime(number):
2     """Return True if 'number' is prime."""
3     for element in range(1, number):
4         if number % element == 0: #Breakpoint
5             return False
6     return True
7
8 is_prime(7)
```

Listing 1: Math Imports

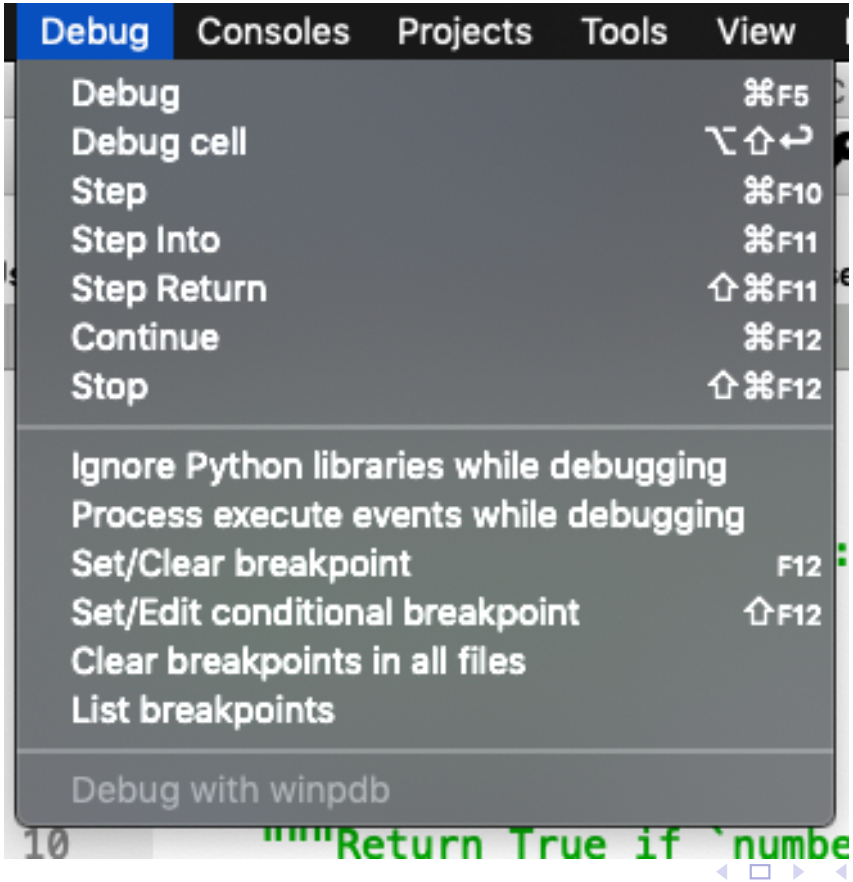
Debugging: Example

In Spyder, we set a breakpoint by choosing the specific line and activating it:

```
 8
 9  def is_prime(number):
10      """Return True if `number` is prime."""
11      for element in range(1, number):
12          if number % element == 0: #Breakpoint
13              return False
14      return True
15
```

Debugging: Example

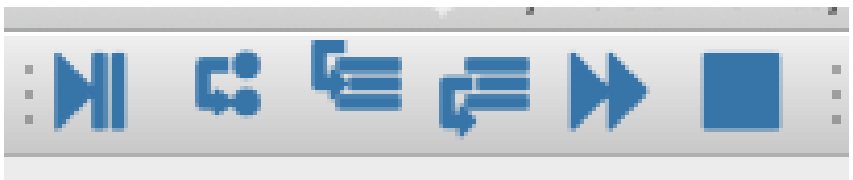
The are two menus to activate the debugging: the main menu and the short menu. The main menu comes from the top and the short is already in the command options. The main menu is:



Debugging: Example

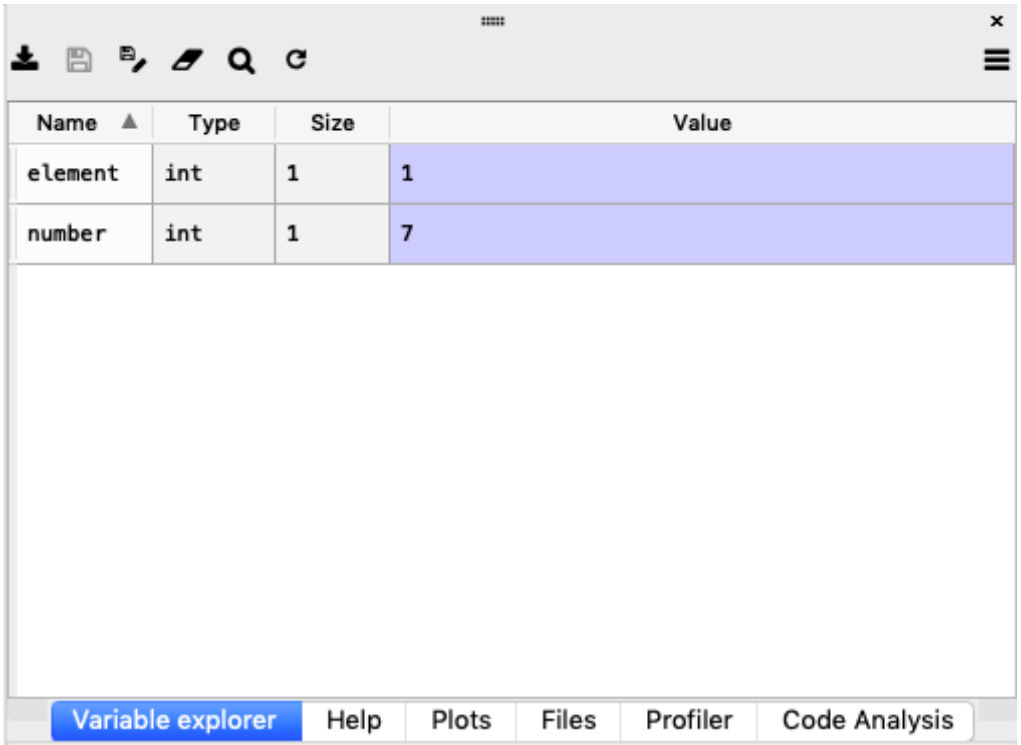
The short menu contains the basic commands, from left to right:

- **Run**: activate debugging.
- **Next**: runs current line.
- **Step**: steps into function.
- **Continue until return**: continues the execution until the next return.
- **Continue**: continues the execution until the next breakpoint.
- **Stop**: ends.



Debugging: Example

We start the debugging process and the program stops in the breakpoint. Now we analyse the program state by looking at the values of the variable in the "Variable Explorer".



Debugging: Example

We can identify an erroneous program state in the example: the comparison will check number % element == 0 with the value 7 and 1, respectively.
Every number module 1 is always o, because 1 can divide any number.
Therefore, 1 should never be checked and the loop need to start in two, instead of 1.

Debugging: Other Examples

Download the following files from the webpage and fix them:

- `leap.py`: checks whether a year is a leap number. The year needs to satisfy to be divisible by 4, but not divisible by 100, with the exception of those divisible by 400.
- `calculator.py`: receives two numbers and calculates their sum, difference, product and division.
- `diamond.py`: receives the number of levels and prints a diamond with '*' symbols up to those levels.

Debugging: Solutions

Download the following files from the webpage and fix them:

- `leap.py`: The error is in line 17, the operation must be the module, not the division.
- `calculator.py`: There is a syntax error in line 24 (extra parenthesis), print errors in lines 26 and 30 and an operation error in line 26 (it must be a product, not exponent).
- `diamond.py`: The prints show extra spaces at the end, they can be removed (lines 29 and 34). The numbers of starts in the first print is wrong, it should be $2 * k - 1$ (line 29). It is also wrong in the second one, the spaces should be $(j + 1)$ and the starts should be $n - 2 * (j - 1)$ (line 34). Also, the condition of the second loop is wrong, it should be $(n - j) >= 1$ (line 33).

Index

Testing

Testing is the process to validate a specific execution of a program.

There are several different ways of perform testing:

- **Unit testing**: tests a specific function (or method) of the program.
- **System testing**: tests the whole input/output process of the program.
- **Regression testing**: tests that the old functionalities of the program remain.
- **Integration testing**: tests that two modules of the program are properly integrated.

We will mainly focus on unit testing.

Testing

It is important to notice that testing can discover bugs in programs, but it can not prove that a program is bug free.

When you create tests for a program you need to thing about the different behaviours of the program and try to activate them.

It is also important to consider some extreme cases, for example, what happen when an input is using the wrong type.

Testing

To activate the testing functionalities of Spyder, we need to install two libraries:

```
conda install -c spyder-ide spyder-unittest
conda install pytest
```

Once, we have done that we can write the tests.

Testing our Prime Numbers function

We are going to create a test for the prime number function. Create a new file called `primes.py`. It will contain the function: `is_prime`:

```
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 2: is_prime function

Testing our Prime Numbers function

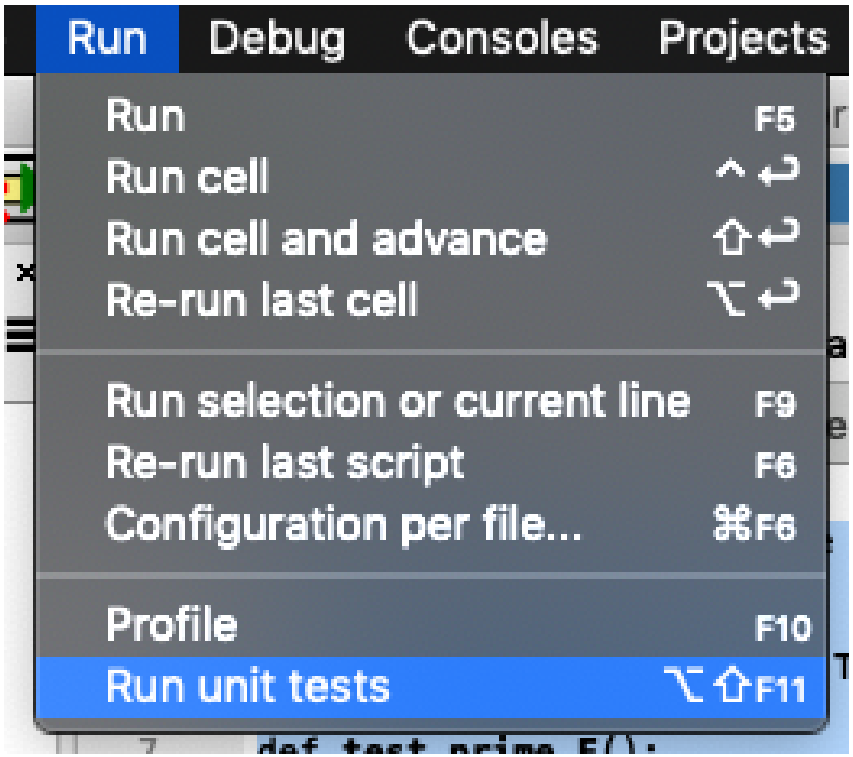
We are going to create a test for the function. Create a new file names `test_primes.py`. The file always need to have the test header followed by the file's name of the file under tests. This new file will contain the tests:

```
1 from primes import is_prime
2
3 def test_prime_T():
4     assert is_prime(11) == True
5
6 def test_prime_F():
7     assert is_prime(8) == False
```

Listing 3: is_prime function

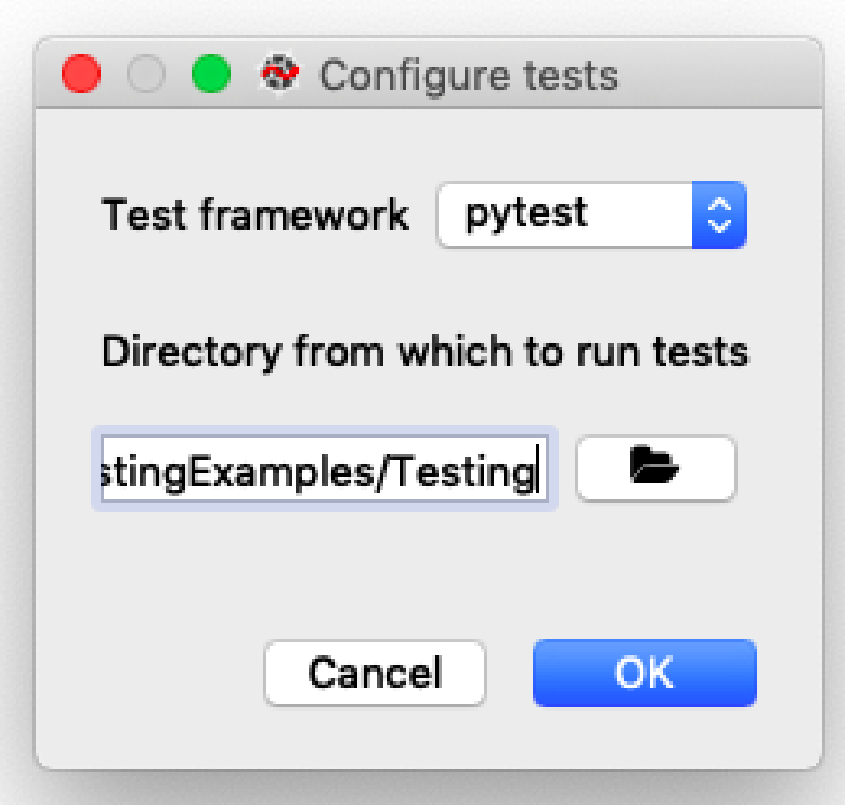
Running the Test

Now, from the "Run" option of Spyder select "Run Unit Test".



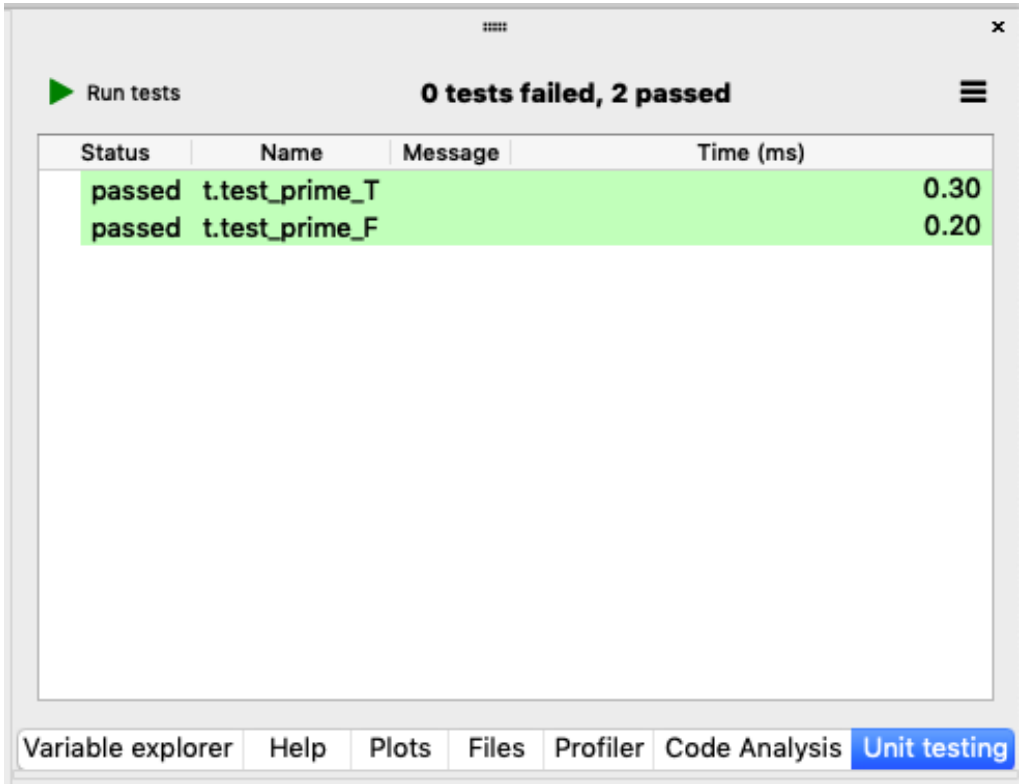
Running the Test

Spyder will ask for the project and testing system: we select pytest and the current project. Then, we press OK and the test will run.



Running the Test

Once the test are done, the results will appear in the right side. It will tell which specific test passed and which ones have not passed, apart of the execution time.



Index

Exercise 1: Testing My Sort Module

Create two tests for every function in the module mysort.py. If you identify any errors on the test, use the debugger to spot the errors.