INDEX PAGE
Pages
Lab 1
FAQ
Lab 2
Lab 3
Lab 7
Lab 6
Testing tips
Lab 4
Lab 10
Lab 5
Module overview
Support
Installing and finding Python
Snippets
Lecture slides
Lab 8
Lab 9
First steps
Testing demo

Laboratory 1: Getting started

Prerequisites: functions, import statements

Contents

- Preparations
- Prerequisites
- Worked example
  - Python prompt notation
  - Python and IPython console -- what is the difference?
- Laboratory 1
- Submit your file

Preparations

1. Start the Spyder environment
2. Reset the name space using the `%reset` command.

This provides us a "clean" IPython interpreter session (within Spyder). You should carry out these steps before every laboratory.

You now have an IPython interpreter available and are encouraged to carry out basic mathematical operations at the iPython prompt (`In [?]:`).

Prerequisites

You should know how to compute simple expressions such as

- `3 + 4`
- `(9 + 3) / 6` and
- `11 * 12`
- Can you work out what `2 ** 2` and `2 ** 0.5` do?

You will need some of these expressions below, and you should be familiar with using the Python Shell to evaluate Python expressions.

You should know how to use a Python development environment. Assuming you use Spyder, you should know how to

- execute Python code in Spyder and
- call functions from the Python prompt.
- For later laboratory sessions, you may also find it useful to know how to change and update function definitions and call the updated function from the Python prompt.

If necessary, go through the exercise First steps executing Python programs interactively which explains how to do this.

Worked example

1. Download lab1.py and save as `lab1.py` (the `1` shows that this file is part of laboratory 1).
2. Open the file with Spyder and try to understand its content.
3. Make sure the function definition `average` is known to Python (by pressing F5, for example) and then check that you can use the `average` function from the Python prompt:

```
In [ ]: average(10, 20)
Out[ ]: 15.0

In [ ]: average(10, 4)
Out[ ]: 7.0
```

```
In [ ]: help(average)
  Help on function average in module __main__:

  average(a, b)
     Given parameters a and b, compute and
     return the arithmetic mean of a and b.
```

Python prompt notation

- Although we expect that you use IPython for these exercises, we will initially also provide examples for the interaction with the Python prompt with `>>>` as displayed by the normal Python console instead of `In [3]:` as displayed by the IPython console: the number 3 here is meaningless as it only counts the number of inputs is that particular session. We write the example above therefore as:

```
>>> average(10, 20)
15.0

>>> average(10, 4)
7.0

>>> help(average)
Help on function average in module __main__:

average(a, b)
      Given parameters a and b, compute and
      return the arithmetic mean of a and b.
```

- Equivalent IPython interaction:

```
In [ ]: average(10, 20)
Out[ ]: 15.0

In [ ]: average(10, 4)
Out[ ]: 7.0

In [ ]: help(average)
Help on function average in module __main__:

average(a, b)
      Given parameters a and b, compute and
      return the arithmetic mean of a and b.
```

Python and IPython console -- what is the difference?

- We also note that we will refer to the IPython console losely as the Python shell or Python console. In terms of the programming language, the IPython and Python shell can process the same command set. The IPython shell is only more powerful in providing tools that help us to write and debug code efficiently.

Laboratory 1

Now, it is your turn. Please define the following functions in the file `lab1.py` and make sure they behave as expected. You also should document them in a way similar to the `average` function.

1. A function `distance(a, b)` that returns the distance between numbers `a` and `b`.

   Examples:

   ```
   >>> distance(3, 4)
   1
   >>> distance(3, 1)
   2
   ```

   Examples (IPython):

   ```
   In [ ]: distance(3, 4)
   Out[ ]: 1
   ```

```
In [ ]: distance(3, 1)
Out[ ]: 2
```

2. A function `geometric_mean(a, b)` that returns the geometric mean of two numbers - i.e. the edge length a square would have to have so that its area equals that of a rectangle with sides `a` and `b`.

   Examples:

   ```
   >>> geometric_mean(2, 2)
   2.0
   >>> geometric_mean(2, 8)
   4.0
   >>> geometric_mean(2, 1)
   1.4142135623730951
   ```

   Examples (IPython):

   ```
   In [ ]: geometric_mean(2, 2)
   Out[ ]: 2.0


   In [ ]: geometric_mean(2, 8)
   Out[ ]: 4.0


   In [ ]: geometric_mean(2, 1)
   Out[ ]: 1.4142135623730951
   ```

3. A function `pyramid_volume(A, h)` that computes and returns the volume of a pyramid with base area `A` and height `h`.

   Example:

   ```
   >>> pyramid_volume(1, 2)
   0.6666666666666666
   ```

   Example (IPython):

   ```
   In [ ]: pyramid_volume(1, 2)
   Out[ ]: 0.6666666666666666
   ```

Submit your file

For every function try to check carefully that your implementation is correct. The tab Testing tips provides some suggestions in form of a short checklist.

Once you have gone through the checklist and think your functions are implemented correctly, please send an email to the coursework submission robot feeg1001@soton.ac.uk with subject `lab 1` and attach the `lab1.py` file.

You should soon get a a confirmation of the submission, and slightly later a report containing feedback through an automated analysis of your code. If the automated analysis says that some test failed, you can change your code and re-submit `lab1.py` by emailing again to the same address. You can repeat this as often as you want.

We urge you to improve and re-submit your work until you pass all tests. Ask the demonstrators for advice to interpret the error messages you may have received in your feedback email if tests have failed.

While the work in this laboratory is not assessed, it is a crucial preparation for the assessment of later laboratory sessions and the class room tests.

Last updated: 2019-01-06 at 17:34

Return to Top