

INTRODUCTION TO PYTHON WITH PANDAS

EXERCISE MANUAL

This page is intentionally blank.

Table of Contents

Exercise 1.1: Introduction to ipython	1
Startup.....	1
As a Simple Calculator.....	1
Introduction to the dir Command.....	4
Executing Shell Commands from Inside ipython.....	4
Magic Commands.....	4
Exit from ipython	4
Exercise 1.2: Simple I/O	6
Exercise 1.3: Editing with ipython	8
Edit File to Make It Look Like the Following	8
Save the Program	9
Running a Python Script.....	9
Exercise 1.4: Debugging with ipython	10
Open ipython.....	10
Execute: %run -d first.py	10

Exercise 1.1: Introduction to `ipython`

Startup

1. Open an Anaconda Prompt or Anaconda PowerShell Prompt, *not* a Command Tool or PowerShell Window.
2. Create a directory and change into it. Your instructor can help with the commands if you are unsure how to do this. We will refer to this as your program directory. If you want, you can work in the Chapter 1 directory of the supplied notebooks.
3. Execute: `ipython`
 - a. You should see:

```
(base) C:\Users\nolan\Google Drive\Python\Introduction to Python With Pandas\programs>ipython
Python 3.8.16 (default, Mar 2 2023, 03:18:16) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.12.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

- b. The `In [1] :` is the prompt.
- c. Previous input commands can be re-executed by entering the command `exec(In[<number>])` where `<number>` is the number associated with the commands

As a Simple Calculator

1. Execute: `3 + 5`
 - a. You should see:

```
In [1]: 3 + 5
Out[1]: 8
```

- b. `Out[1]` is the list of outputs.
2. Execute: `10 / 3`

- a. You should see:

```
In [2]: 10 / 3
Out[2]: 3.3333333333333335
```

- b. All arithmetic is done with double float.
- c. The 5 at the end of output cannot be trusted.

3. Execute: `cos(2 * pi)`

a. You should see:

```
In [3]: cos(2 * pi)
-----
NameError                                Traceback (most recent call last)
Cell In[3], line 1
----> 1 cos(2 * pi)

NameError: name 'cos' is not defined
```

b. The standard C language maths functions are in a separate file, called a module.

c. To use functions, you have to give the interpreter access to the module. This is called importing and is done with the `import` statement.

4. Execute: `import math`

5. Now try:

a. You should see:

```
In [5]: math.cos(2 * math.pi)
Out[5]: 1.0
```

6. To find out a little about the module, execute `math?`

a. You should see:

```
In [6]: math?
Type:      module
String form: <module 'math' (built-in)>
Docstring:
This module provides access to the mathematical functions
defined by the C standard.
```

b. The `Docstring` provides some documentation on the module.

i. The command line command `pdoc <object>` returns just the docstring associated with the object.

ii. Inside of `ipython`, use `help(<object>)`

c. The command `<object>??` may give you more information.

7. Execute: `help(math)`

a. See:

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.
-- More --
```

b. Exit with a `q`.

8. Execute: `help(math.cos)`

a. The imported module creates a new namespace. Thus, to access the `cos` function, you use `math.cos`; to access the value of `pi` you use `math.pi`.

b. See:

```
In [8]: help(math.cos)
Help on built-in function cos in module math:

cos(x, /)
    Return the cosine of x (measured in radians).
```

c. The final `/` signifies the end of the *positional only* parameters, parameters you *cannot* use as keyword parameters. We will see more on this later.

Introduction to the `dir` Command

1. Execute: `dir(math)`
 - a. For a module returns a list of the module's attributes; some of which are executable functions of the module.

Executing Shell Commands from Inside `ipython`

To execute a shell command from within `ipython`, precede the shell command with an exclamation point.

1. On Windows try `!dir`
2. On Linux try `!ls`

Magic Commands

Magic commands are commands executed by `ipython`. Their format is `%<command>`. We will see several of these commands while working through this chapter.

1. Execute: `%ls`
 - a. This will list the directory contents. The magic commands that interact with the shell usually follow the Linux conventions.
2. To see a list of all magic commands, execute: `%lsmagic`
3. To see documentation on a magic command, execute: `<command>?`

Exit from `ipython`

1. Execute:
`exit()`

This page is intentionally blank.

Exercise 1.2: Simple I/O

1. Log in and open `ipython`.
2. Ask for a person's name to be input from `stdin`.
 - a. Assign the object input to the variable `name`.
 - b. Write below how you will do this in python in `ipython`.

3. Write out the string `"hello, <name>"`.
 - a. Use the `name` from Step 2.
 - b. Write below how you will do this in python in `ipython`.

4. Exit from `ipython`.

This page is intentionally blank.

Exercise 1.3: Editing with `ipython`

Use whichever text editor you are familiar with.

Edit File to Make It Look Like the Following

```
1  #!/usr/bin/env python3
2
3  """
4      filename: first.py
5  """
6
7  name = input("What is your name? ")
8  print("Well, hello", name)
```

1. Line 1:
 - a. This is called the interpreter line. It has no effect on Windows but does not harm and makes the script runnable on Linux.
 - b. `#!`, the magic number, tells the shell this file is interpreted by the program that follows.
 - c. `/usr/bin/env` tells the shell to look for the interpreter using the `PATH` environment variable.
 - d. `python3` is the version of python wanted. On many systems, you would use `python`.
 - e. The script is executed by entering `python <script_name>` on the command line.
2. Lines 3 through 5:
 - a. This is called the docstring. It must be the first entry in a Python script after the interpreter line.
 - b. A docstring starts with 3 double or single quotes and ends with 3 matching quotes. Spaces and newlines have meaning inside the docstring.
 - c. This is used in the Python `help` and `ipython ?` documentation systems.

-
3. Lines 2 and 6:
 - a. Blank lines have no meaning.
 4. Spacing at the beginning of the line is very important; it specifies the structure of the program.
 - a. The docstring must be the first thing on the line. *Note:* material inside of the docstring is at the programmers' whim.
 - b. The executable lines must be at the beginning of the line. Much, much more in chapter on program structure and chapter on classes.

Save the Program

Use the rules of your editor. If you don't want to create a file, there is one in the Chapter 1 directory.

Running a Python Script

1. Make sure you are in the right directory.
2. Execute: `python ./first.py`
 - a. This should run the program.

Exercise 1.4: Debugging with `ipython`

Open `ipython`

1. If necessary, log in, open `ipython`, and change directory to the directory where you created `first.py`.

Execute: `%run -d first.py`

1. The `-d` is to start the debugger.
2. You should see:

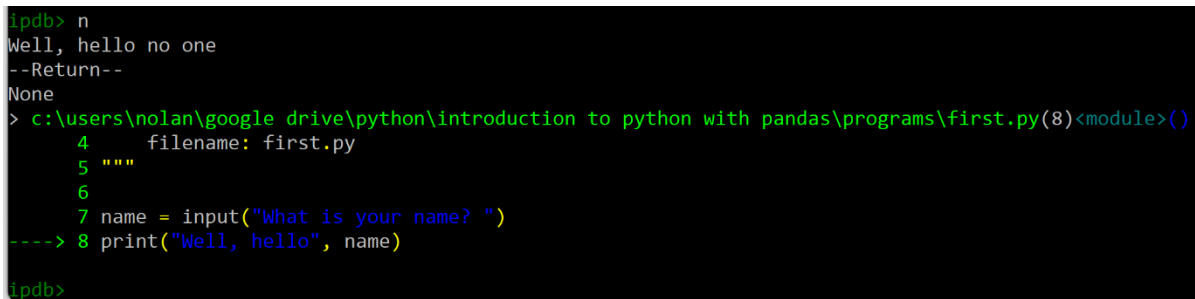
```
In [1]: run -d first.py
*** Blank or comment
*** Blank or comment
*** Blank or comment
*** Blank or comment
NOTE: Enter 'c' at the ipdb> prompt to continue execution.
> c:\users\nolan\google drive\python\introduction to python with pandas\programs\first.py(3)<module>()
   1 #! /usr/bin/env python3
   2
----> 3 """
      4     filename: first.py
      5 """
ipdb>
```

- a. The green arrow points to the next command to be executed. NOTICE that this is at the docstring.
3. Tap `n` or enter `Next`.
 - a. You should see:

```
ipdb> n
> c:\users\nolan\google drive\python\introduction to python with pandas\programs\first.py(7)<module>()
   4     filename: first.py
   5 """
   6
----> 7 name = input("What is your name? ")
   8 print("Well, hello", name)
ipdb>
```

- b. The green arrow now points to line 7, the next command to be executed.

-
4. Enter: `print(name)`
 - a. It will give an error because the command has not been executed and `name` has not been created.
 5. Enter: `n`
 - a. You should see the prompt to enter a name.
 - b. The command will not finish until a new-line character (`\n`) is entered through `stdin`.
 6. Enter a value.
 - a. The green arrow will move on.
 7. Enter: `print(name)`
 - a. This time the command will work and print the value you entered.
 8. Enter: `name='no one'`
 - a. `'no one'` is a string. A string can be enclosed in double or single quotes.
 9. Enter: `n`
 - a. You should see:



```
ipdb> n
Well, hello no one
--Return--
None
> c:\users\nolan\google drive\python\introduction to python with pandas\programs\first.py(8)<module>()
4     filename: first.py
5     """
6
7     name = input("What is your name? ")
----> 8     print("Well, hello", name)
ipdb>
```

- b. The print command has executed. See the `"Well, hello no one"`. Note we changed the value of the variable earlier.
- c. `None` is the return value. If run at the command line, this would be translated to a 0 return code.
- d. The script has terminated. Unfortunately, the debugger under `ipython` doesn't clean up as well as it should. Enter: `<Ctrl-D>` to exit the debugger.