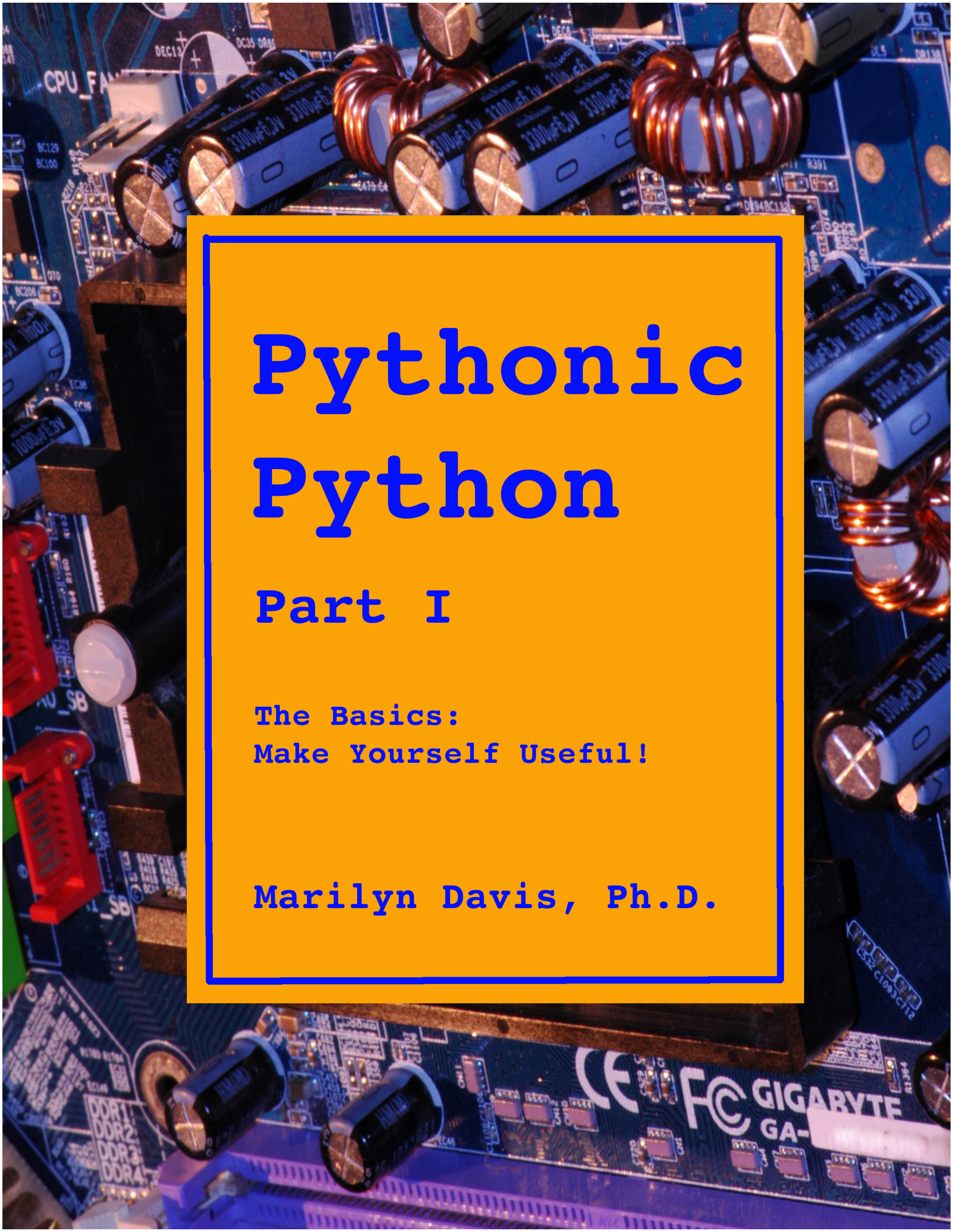


Pythonic Python

Part I

The Basics:
Make Yourself Useful!

Marilyn Davis, Ph.D.

Pythonic Python

Marilyn Davis, Ph.D.
Your Instructor

Cover images by ClintDavis.Com

Lab 1 – Birds Eye View

- Executing a Python program
- Writing to `stdout`
- Assigning: labels and objects
- `strings`
- Idle

Lab 2 – Branching and Looping

- `if`, `elif`, and `else`
- `while` and another `else`
- Iterating with a `for` loop
- Counting loop with `range`
- Relational and logical operators
- `tuples`

Lab 3 – Input and Exceptions

- Input from `stdin`
- Factory functions
- Catching an exception:
-     yet another `else`

Lab 4 – Formatting Strings

- Formatting strings
- Integer division issue

Lab 5 – Functions

- Function protocols

Lab 6 – import

- `import`
- Module: `random`
- Introspection

Lab 7 – Attribute Scope

- Indentifier scope

Lab 8 – Flexible Functions

- Default function arguments
- Keyword function arguments

Lab 9 – Sequence Slicing

- Sequences:
-     Types: `str`, `tuple`, `list`
-     Accessing elements
-     Slicing

Lab 10 – Sequence Accumulating

- Accumulating Sequences

Lab 11 – Sequence Differences

- Sequence Differences
- String Manipulation

Lab 12 – list Facilities

- Sorting Sequences
- `str` functions

Lab 13 – Sequences And Mutability

- Sequence Mutability

Lab 14 – sys Library

- Module: `sys`

Acknowledgements

These materials are the product of hundreds of programmers in the Silicon Valley. I'm grateful for their feedback: finding errors, making suggestions, and honest reactions.

Thanks are also due to my family, Clint and Charlie Davis, for being proud of me, and relentlessly encouraging me to do what I really want to do, in spite of external pressures.

How To Use This Udemy Course

This online course is designed to emulate the face-to-face class that I developed while helping hundreds of engineers in the Silicon Valley, and beyond, to learn Python.

It is expected that you are already a programmer and want to learn to program with Python. There are no explanations in this material of the basic concepts of software engineering. You are expected to already know these.

My goal in developing this class is to enable you to learn Python quickly, and while having fun. You'll find that Python development, itself, is just like that: quick and fun!

I hope that you'll find the material presented in *curiosity order* and that the questions in your mind are answered quickly.

This written material has very little written explanation. It is mostly program examples, exercises, and solutions; it motivates the video lectures.

There are two versions of this pdf: one for printing, and one for viewing.

I recommend that you print, onto paper, the *for printing* pdf, double-sided, 3-hole-punched. Put this pdf into a 1.5 inch, 3-ring binder.

The first page is a *material map*, which shows that the solutions to the exercises are at the back of the pdf.

There is a *break* page, page 77-78 of the pdf. Put a tag on that page so that you can find it easily.

Your Python study will start with page 9 of this printed pdf. Follow along with the discussion in the video *Lab 01:Birds Eye View*, taking copious notes on your printed pdf; then work through the exercises that start on Lab 1, page 4 (or page 12 of the pdf).

When you have finished with the Lab 1 exercises, go to your tagged break page to find the solution to the exercise, which is Lab 1, page 7. Place that page into the binder after page Lab 1, page 6 and follow the video *Lab 01:Solutions*.

Then we'll move on to *Lab 02:Branching and Looping*.

In the end, all the pages will fit together, and there is an index. I hope you find the notebook you assemble and annotate to be a valuable reference in your work with Python.

Thank you for taking this course with me. I hope you enjoy it; I know you will enjoy Python.

Marilyn Davis

Instructor: Marilyn Davis, Ph.D.

Email Address: marilyn@pythontrainer.com

Phone Number: (650) 814-4435

Book Recommendation

The Quick Python Book Second Edition by Vernon L. Ceder; ISBN # 9781935182207, published by Manning.

Regrettably, it's about Python 3, but the two Python languages are similar enough that you can learn about both Pythons from this excellent book.

Online Resources

1. `http://www.python.org` and in particular: `http://www.python.org/doc` has very helpful documentation, online and free.
2. *The Python Cookbook* by Alex Martelli, Anna Martelli Ravenscroft & David Asher. This is a very interesting collection of Python code, best read after you have taken the class. ISBN # 0-596-00797-3 and online free.

Python 2 Interpreter Please go to `http://python.org/download/` to download Python 2.7 for your computer. The Idle development environment comes in with the interpreter and, unless you have another environment you are using, this is a good way to start.

Why not Python 3?

Please see `http://wiki.python.org/moin/Python2orPython3`. In my experience, by far most engineers are interested in Python 2 to continue the work of their companies. While the move from another language to Python brings big gains in productivity, readability, and reusability, moving from 2 to 3 has no such gains. Also, because some of the libraries have not yet been ported to 3, I believe that Python 2 is the practical choice for now.

Lab 01

Birds Eye View

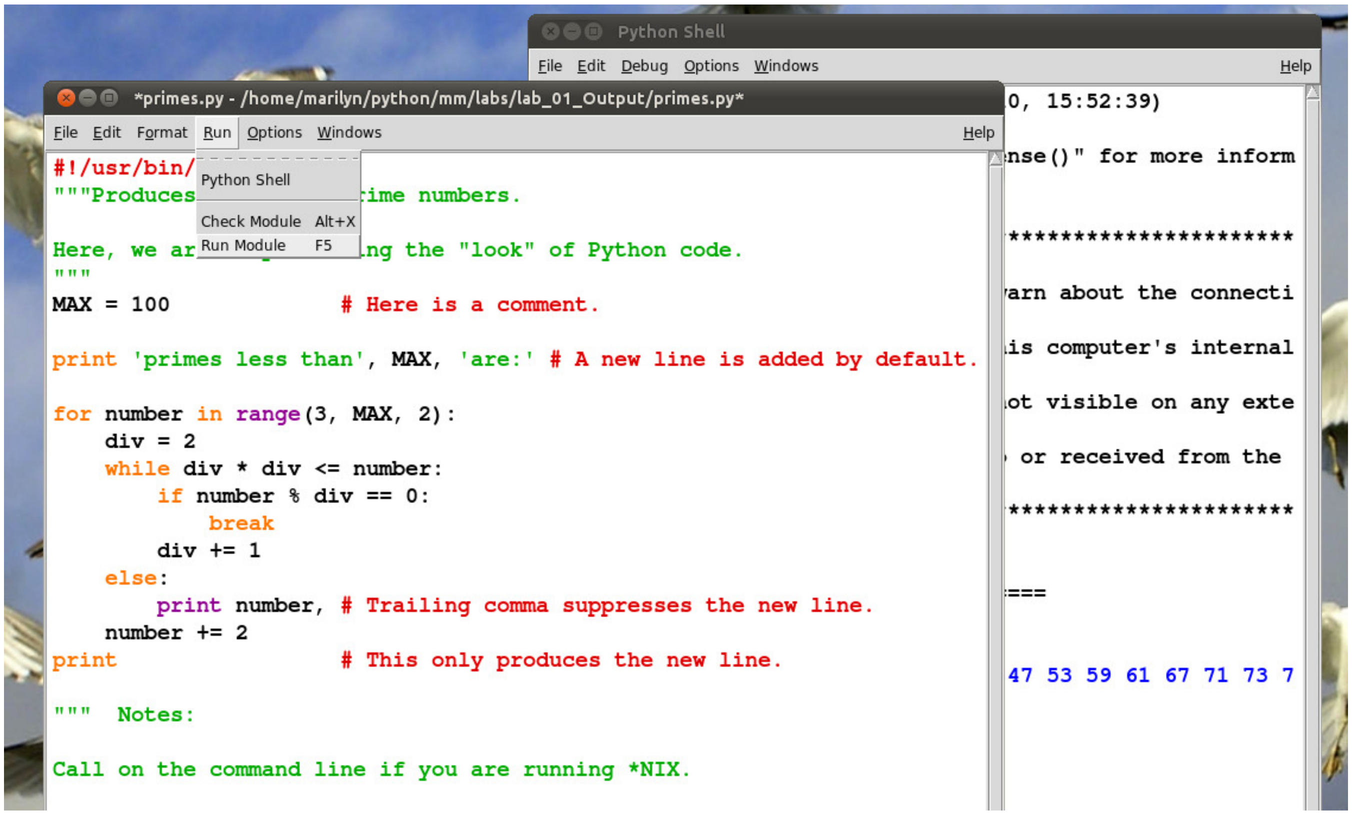
- Executing a Python program
- Writing to stdout
- Assigning: labels and objects
- strings
- Idle

primes.py

```python
#!/usr/bin/env python
"""Produces a list of prime numbers.

Here, we are only checking the "look" of Python code.
"""
MAX = 100                  # Here is a comment.

print 'primes less than', MAX, 'are:' # A new line is added
                                      # by default.

for number in range(3, MAX, 2):
    div = 2
    while div * div <= number:
        if number % div == 0:
            break
        div += 1
    else:                  # Overloaded 'else', loop didn't 'break'.
        print number,      # Trailing comma suppresses the new line.
print                      # This only produces the new line.
```

```
Notes:

Call on the command line if you are running *NIX.

$ primes.py
primes less than 100 are:
3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97


Or, invoke the interpreter:

$ python primes.py
primes less than 100 are:
3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

```
Or, ask the interpreter to run it and then stay active for
introspection.

$ python -i primes.py
primes less than 100 are:
3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
>>> number
99
>>>
```


Python Shell
File Edit Debug Options Windows Help
primes.py - /home/marilyn/python/mm/labs/lab_01_Output/primes.py
File Edit Format Run Options Windows Help
Python Shell
Check Module Alt+X
Run Module F5

```
#!/usr/bin/
"""Produces              ime numbers.

Here, we ar             ng the "look" of Python code.
"""
MAX = 100               # Here is a comment.

print 'primes less than', MAX, 'are:' # A new line is added by default.

for number in range(3, MAX, 2):
    div = 2
    while div * div <= number:
        if number % div == 0:
            break
        div += 1
    else:
        print number, # Trailing comma suppresses the new line.
    number += 2
print                   # This only produces the new line.

"""  Notes:

Call on the command line if you are running *NIX.
```

```
0, 15:52:39)
nse()" for more inform
**********************
arn about the connecti
is computer's internal
ot visible on any exte
 or received from the
**********************
===
47 53 59 61 67 71 73 7
```

Figure 1: Idle Development Environment

Integrated development environments for Python abound. We will be using *idle*, the original Python environment, because it is free and a no-brainer. But, some others are much better:

- `http://stackoverflow.com/questions/81584/what-ide-to-use-for-python`
- `http://wiki.python.org/moin/IntegratedDevelopmentEnvironments`

output.py

```
#!/usr/bin/env python
"""Demonstrates 4 ways to delimit strings."""

print 'Hello world'
print
print 'She said "Hello world"'
print
print "She said 'Hello world'"
print
print '''Little dark woman of my suffering,
with eyes of flying paper,
you say "Yes" to everyone,
but you never say when.
'''  # end of string started on line 10.
```

```
$ output.py
Hello world

She said "Hello world"

She said 'Hello world'

Little dark woman of my suffering,
with eyes of flying paper,
you say "Yes" to everyone,
but you never say when.

$
```

Raw strings:

```
>>> print r"\n"
\n
```

These come in handy with regular expressions.

.

Lab 01 – Exercises:

1. The interpreter works as a calculator. Type this at the prompt:

   ```
   >>> 1 + 2 + 4
   ```

   Then try:

   ```
   >>> _ + 8
   ```

   Type:

   - On Windows and Linux:
     - **Alt-p** repeatedly to cycle through previous lines.
     - **Alt-n** repeatedly to cycle back.
   - On Mac:
     - **Ctrl-Alt-p** repeatedly to cycle through previous lines.
     - **Ctrl-Alt-n** repeatedly to cycle back.

   Now, make a program *module* (script) with the `1 + 2 + 4` line in it, i.e., `1 + 2 + 4`. Run it. Does it work? No? Fix it.

   Note that the interpreter displays its evaluation of the line but that a program will not display unless you ask it to `print`.

2. Although Python tries to remove the back-slash from your code. Sometimes you need it.

   Try this in the interpreter:

   ```
   >>> greeting = "Hello \nworld."
   >>> greeting
   ```

   Now try:

   ```
   >>> print greeting
   ```

   Notice that `print`ing interprets the backslash-`n` to create a new line, while evaluating just spits out the raw string.

   And try:

   ```
   >>> food = "popcorn"
   >>> print food * 3
   ```

   and

   ```
   >>> food + food
   ```

   + means **concatonation** to strings.

   ∗ means **repetition** to strings.

3. Write a script to produce this output — EXACTLY :

   ```
             He said "Hello World".
             She said 'Hello Sky'.
             She said "He said 'Hello World'".
   ```

   (Hint: This is very easy if you remember to use triple-quotes. In fact, try to always avoid using a backslash in your code to keep it *Pythonic*, i.e., readable.)