PYTHON LAB BOOK

Python For Programmers

UCSC Extension Online

Marilyn Davis, Ph.D.
Your Instructor

Instructor: Marilyn Davis, Ph.D.

Email Address: marilyn@pythontrainer.com

Phone Number: (650) 814-4435

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.

If you want to use Python 3, that's ok too. These materials are about Python 2; any books you buy are probably about Python 3. I accept your assignments in either Python version.

Suggested Book

The Quick Python Book, 2nd Edition, Naomi R. Ceder, Manning Publications, 2010, ISBN-10: 193518220X, ISBN-13: 978-1935182207.

Python Class Style Guide

One big advantage of Python is its readability. When you write your code, always think of it as literature or art, something to be admired by a reader.

If you don't understand a style listed below, we have probably not yet learned that part of Python.

1. You may break the style guide in certain circumstances – but you must have a good reason for everything you do that does not follow the style guide. Document that reason!

2. Do what is expected. Don't give your reader a reason to think about anything but the intention of the code. No distractions; nothing unnecessary.

3. Be tasteful about documentation:

   - Your code should not include documentation about language features, or about anything obvious.
   - Do provide complete documentation about your modules, functions, and classes in doc strings.
   - Use the `#` when you want to document details for the reader of the code.
   - There should be no external documentation, i.e., no `README` files.

4. Indentations are 4 spaces. Don't create deep nests of indents. Instead, improve your architecture. Maybe make more functions.

5. No duplicate code (after you have learned to make functions).

6. All functionality should be in functions (after you have learned to make functions).

7. Labels are meaningful, long if necessary, and self-documenting. The case of your labels means a lot to the reader:

   - `variable_labels` are lower case with an underline to separate words.
   - `CONSTANT_LABELS` are all upper case.
   - `FunctionLabels` have every word capitalized, including the first. They start with verbs or are verbs.
   - `ClassName` labels have every word capitalized, including the first. They are nouns.
   - `module_labels.py` are lower case with an underline to separate words.

8. A comma has a space after it, and no spaces before it.
9. Usually the assignment operator has one space on each side:

   ```
   bacon_strips = 3
   ```

   But:

   - In function definitions, there are no spaces around the = in defaulted arguments:

     ```
     def RateIt(name, rating=100):
       pass
     ```

   - In function calls, when providing keyword arguments, there are no spaces around the =:

     ```
     RateIt("Alice's", rating=99)
     ```

10. Parentheses have no spaces around them. Also:

    - No extra parentheses, unless they help readability.
    - Conditionals don't have parentheses:

      ```
      if chocolate_candy == "dark":
          pass
      ```

Grading

Your grade will be the average of the scores you get on 8 **Reviews** that are available by clicking in the left menu on **Test & Quizzes**, under **COURSE TOOLS**.

Each Review has a due-date, but it is a *suggested* due-date so that you know if your pace is good for completing the course on time. There is no penalty for being late.

The Reviews are not Tests or Quizzes, but learning activities. You are welcome to research the answers any way you wish.

In particular, I hope that you will raise any doubts you have about the correct answers in the **Forum** so we can all discuss the questions.

You can only submit each Review once, i.e., no re-submissions, so try to get the answers correct before you submit your Review.

You can **SAVE** your Review many times before submitting it. Just be sure you like your answers before you **SUBMIT**.

Assignments

You'll find that there are 6 **Assignments**. ***These are not used to calculate your grade.***

However, to get the most from this class, and to become the best programmer that you can be, you do want to do the Assignments and submit them for critique.

You'll see that, although there are 6 Assignments listed, in reality, there are only 3. You have two opportunities to submit each: a ***Draft*** submission, and a ***Final*** submission.

After you submit your draft, I'll critique it and return it and my solution will become available to you.

Warning: You ***must*** follow the class style guide, **Style Guide For Assignments** tab on the left. If your code takes extra time for me to read because of meaningless identifiers, confusing capitalizations, etc., I'll simply return it and you can to try again on your Final submission.

Also, you must turn your assignments in on time so that I can plan my schedule, and keep my work load consistent.

These **Assignments** appear as homework assignments in Labs 7, 11, and 17. That's when you should tackle them.

Lab 1 – Output

- Executing a Python program.
- Syntax: code blocks, colons.
- `if`, `elif` and `else`
- `while` and another `else`
- Writing to `stdout`
- Relational and logical operators

Lab 2 – Input

- Input from `stdin`
- Factory functions
- Catching an exception:
- yet another `else`
- Formatted strings
- Integer division issue

Lab 3 – for range

- `range` operator
- `for` loop
- tuples

Lab 4 – Functions

- Function protocols
- `import` and `reload`
- Module: `random`
- Introspection

Lab 5 – Scope

- Indentifier scope
- Default arguments
- Keyword arguments

Lab 6 – Sequences

- Sequence types: `str`, `tuple`, `list`
- Sequence slicing and other manipulations

Lab 7 – Important Trick

- Module: `sys`
- Important trick:
- `__name__` and `'__main__'`
- Valid identifiers

Lab 8 – Comprehensions

- Scope issues
- List comprehensions

Lab 9 – Dictionaries

- Importing with `from`
- Dictionaries

Lab 10 – File IO

- File I/O
- Module: `os`
- Walking A Directory

Lab 11 – Packages

- Modules: `shutil`, `tempfile`
- Python Packages

Lab 12 – Dynamic Code

- Dynamic Code Generation
- Modules:
- `subprocess`
- `glob`
- `profile`

Lab 13 – Function Fancies

- Function protocols: variable length argument lists
- Formatted printing using a dictionary for replacement
- Unpacking sequences and dictionaries
- Generators (Optional)
- Decorators (Optional)

Lab 14 – OOP

- Module: `shelve`
- Classes
- Inheritance
- Class variable

Lab 15 – Overriding

- Overriding
- *Has-A* vs *Is-A* relationships

Lab 16 – New Style Classes

- Useful attributes
- Iterators
- New style classes
- Attribute control (Optional)
- `property` (Optional)
- Static methods (Optional)
- Class methods (Optional)
- Diamond inheritance (Optional)

Lab 17 – Developer Modules

- Context Manager class
- Module: `unittest`
- Module: `optparse`

Lab 18 – Wrap Up

- Exceptions
- Namespaces
- Nests
- Pitfalls
- Finding Modules and Help

Lab 19 – re Module

- re - Regular Expressions (Optional)
- Search and replace (Optional)
- Named groups (Optional)

Lab 20 – re Syntax

- Regular expression syntax (Optional)
- Testing regular expressions (Optional)

PYTHON LAB BOOK

Python For Programmers
UCSC Extension Online

Lab 1 Output

Topics

- Executing a Python program.
- Syntax: code blocks, colons.
- `if`, `elif` and `else`
- `while` and another `else`
- Writing to `stdout`
- Relational and logical operators

v.4.0

primes.py

```
 1 #!/usr/bin/env python
 2 """primes.py -- Produces a list of prime numbers.
 3 Here, we are only checking the "look" of Python code.
 4 """
 5 MAX = 100              # Here is a comment.
 6
 7 print 'primes are: '  # A new line is added by default.
 8 number = 3
 9 while number < MAX:
10     div = 2
11     while div * div <= number:
12         if number % div == 0:
13             break
14         div += 1
15     else:              # Overloaded 'else', loop didn't 'break'.
16         print number, # Trailing comma suppresses the new line.
17     number += 2
18 print                 # This only produces the new line.
19
20 """  Notes:
21
22 Call on the command line if you are running *NIX.
23
24 $ primes.py
25 primes are:
26 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
27
28 Or, invoke the interpreter:
29
30 $ python primes.py
31 primes are:
32 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
33
34 Or, ask the interpreter to run it and then stay active for
35 introspection.
36
37 $ python -i primes.py
38 primes are:
39 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
40 >>> number
41 101
42 >>>"""
```

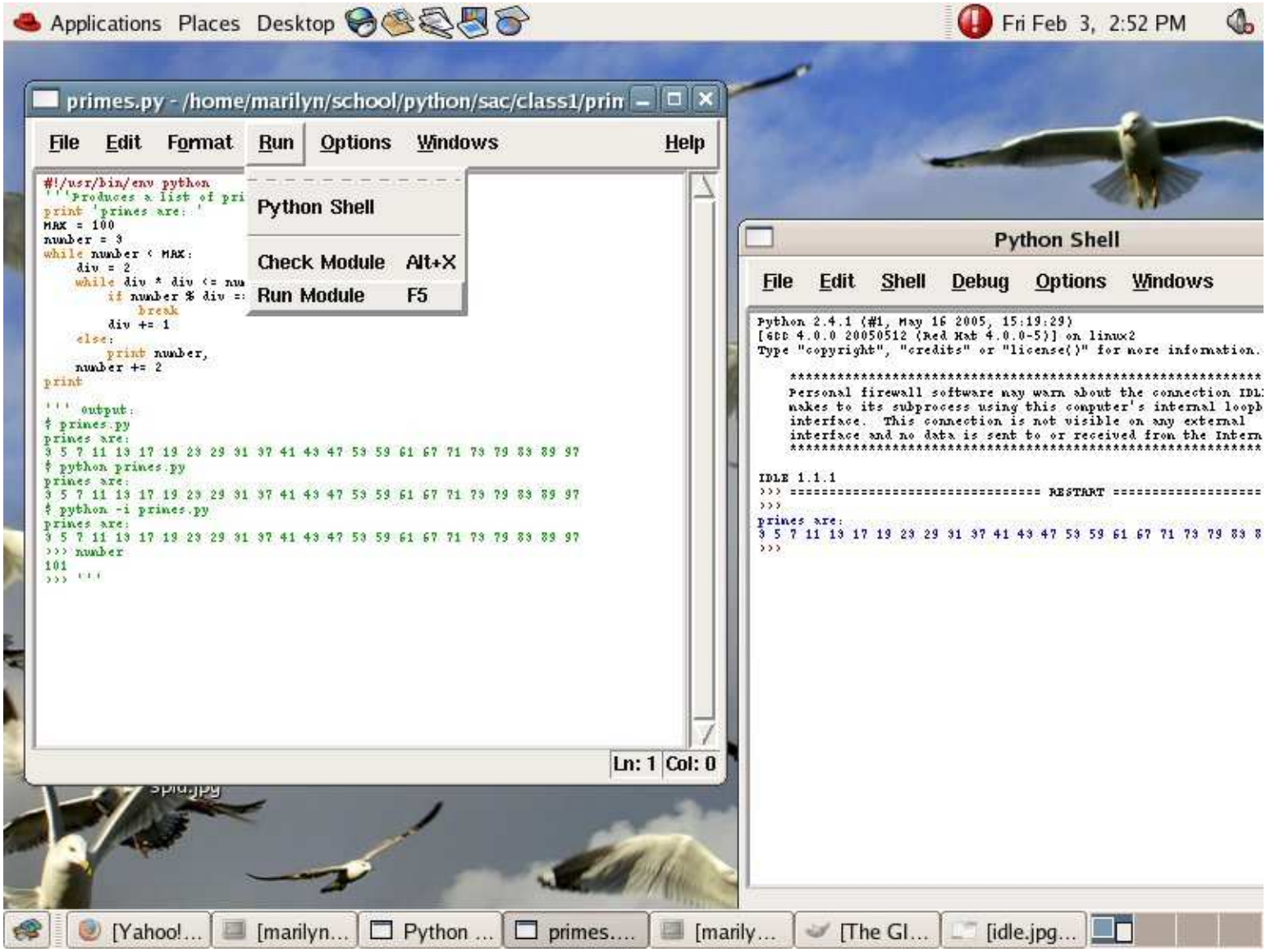


Figure 1: Idle Development Environment

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://spyced.blogspot.com/2005/09/review-of-6-python-ides.html`
`http://wiki.python.org/moin/IntegratedDevelopmentEnvironments`

output.py

```
 1 #!/usr/bin/env python
 2 """output.py  Demonstrates 3 ways to delimit strings."""
 3
 4 print 'Hello world'
 5 print
 6 print 'She said "Hello world"'
 7 print
 8 print "She said 'Hello world'"
 9 print
10 print """Little dark woman of my suffering,
11 with eyes of flying paper,
12 you say "Yes" to everyone,
13 but you never say when.
14 """  # end of string started on line 10.
15
16 """  # An unlabeled string is a comment.
17 $ output.py
18 Hello world
19
20 She said "Hello world"
21
22 She said 'Hello world'
23
24 Little dark woman of my suffering,
25 with eyes of flying paper,
26 you say "Yes" to everyone,
27 but you never say when.
28
29 $
30 ----
31
32 Raw strings:
33 >>> print r"\n"
34 \n
35
36 These come in handy with regular expressions.
37 """
```

conditions.py

```
 1 #!/usr/bin/env python
 2 """conditions.py demonstrates if/elif/else and while/else."""
 3
 4 number = 4
 5
 6 # if/elif/else
 7
 8 if number < 10:
 9     print number, 'is small.'
10 elif number >= 1000:
11     print number, 'is big.'
12 else:
13     print number, 'is medium.'
14
15 # Alternate syntax for 2.5 -- all one line but less readable.
16
17 print number, "is",
18
19 print "small." if number < 10 \
20                 else "big." if number >= 1000 \
21                 else "medium."
22
23 # else occurs in a loop too
24
25 while number < 6:
26     if number % 3 == 0:
27         print number, 'is divisible by 3.'
28         break
29     number += 1
30 else:
31     print 'Nothing in the loop was divisible by 3.'
32
33 """
34 $ conditions.py
35 4 is small.
36 4 is small.
37 Nothing in the loop was divisible by 3.
38 $"""
```

Relational Operators in Python:

$<$	means less than
$>$	means greater than
$<=$	means less than or equal
$>=$	means greater than or equal
$==$	means equal
$!=$	means not equal

Logical Operators:

`and`	means and
`or`	means or
`not`	means not

Lab 01

1. (Adapted from Chun 2-2) The interpreter works as a calculator. Type this at the prompt:

```
1 + 2 + 4
```

   Now, make a script with the same line in it. Does it work as you expect? No? Fix it.

2. 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:

```
>>> print greeting * 3
```

   and

```
>>> print greeting + '\nHello.'
```

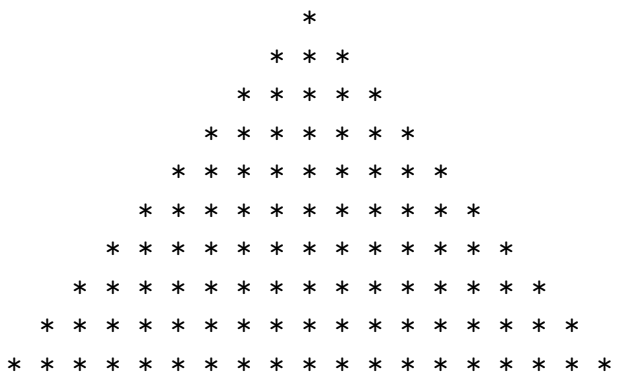
3. Write a program to produce this output — EXACTLY :

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

4. Write a script that uses a while loop to produce this output:

```
 10 9 8 7 6 5 4 3 2 1 BLASTOFF!!!
```

5. (Optional) Write a script that uses nested while loops to produce this pattern:



Can you find an easier way? Hint: Have another look at exercise 2.

Index