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) 965-7121

Some Books

1. *Dive Into Python* by Mark Pilgrim, online at: `http://diveintopython.org`

   This book, free and online, gets rave reviews.

2. *Core Python Programming, Second Edition* by Wesley J. Chun; ISBN # 0-13-226993-7

   This excellent, comprehensive book is for people who already know how to program. We will use some exercises and examples in Chun's book.

3. *Learning Python, Second Edition* by Mark Lutz & David Ascher; ISBN # 0-596-00281-5.

   Excellent book, especially for people who already know how to program. Class notes borrow heavily from this book.

4. `http://www.python.org` and in particular: `http://www.python.org/doc` has very helpful documentation, online and free.

5. *The Python Cookbook, Second Edition* by Alex Martelli, Anna Martelli Ravenscroft & David Asher; ISBN # 0-596-00797-3

   Excellent read, after you know Python. It helps you get the knack of Pythonic thinking.

6. *Python How to Program* by Deitel, Deitel, Liperi and Wiederman; ISBN # 0-13-092361-3.

   We use a few exercises from this book.

Grading

You have two choices for how you earn your grade. Choose **one**:

1. 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.

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

   In particular, we hope that you will raise any doubts you have about the correct answers in the **Forums** 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.

2. Your grade will be the average of 6 scores. You earn your 6 scores by completing 3 **Assignments**, also under **COURSE TOOLS** in the left menu. For each Assignment:

   - You submit your Assignment solution: a ***draft***.
   - We give you one score; and a model solution for the Assignment.
   - You improve your Assignment, and submit your ***final*** solution.
   - You receive a second score for the same problem.

   The Assignment specifications also appear in Labs 7, 11, and 17 because that is when you are ready to tackle them.

Other Resources

In the left menu, under **Resources**, you'll also find `labs.zip` and `code.zip`:

- **labs.zip** has partially solved exercises, to keep you focused on the new material. This becomes important later in the class.
- **code.zip** has all the class code in it, except solutions to the Assignments. You will receive these solutions after you submit your draft solutions.

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

```
#!/usr/bin/env python
"""primes.py -- Produces a list of prime numbers.
Here, we are only checking the "look" of Python code.
"""

MAX = 100             # Here is a comment.

print 'primes are: '  # A new line is added by default.
number = 3
while number < MAX:
    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.
    number += 2
print                 # This only produces the new line.

"""
Notes:

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

$ primes.py
primes 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 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 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
101
>>>"""
```

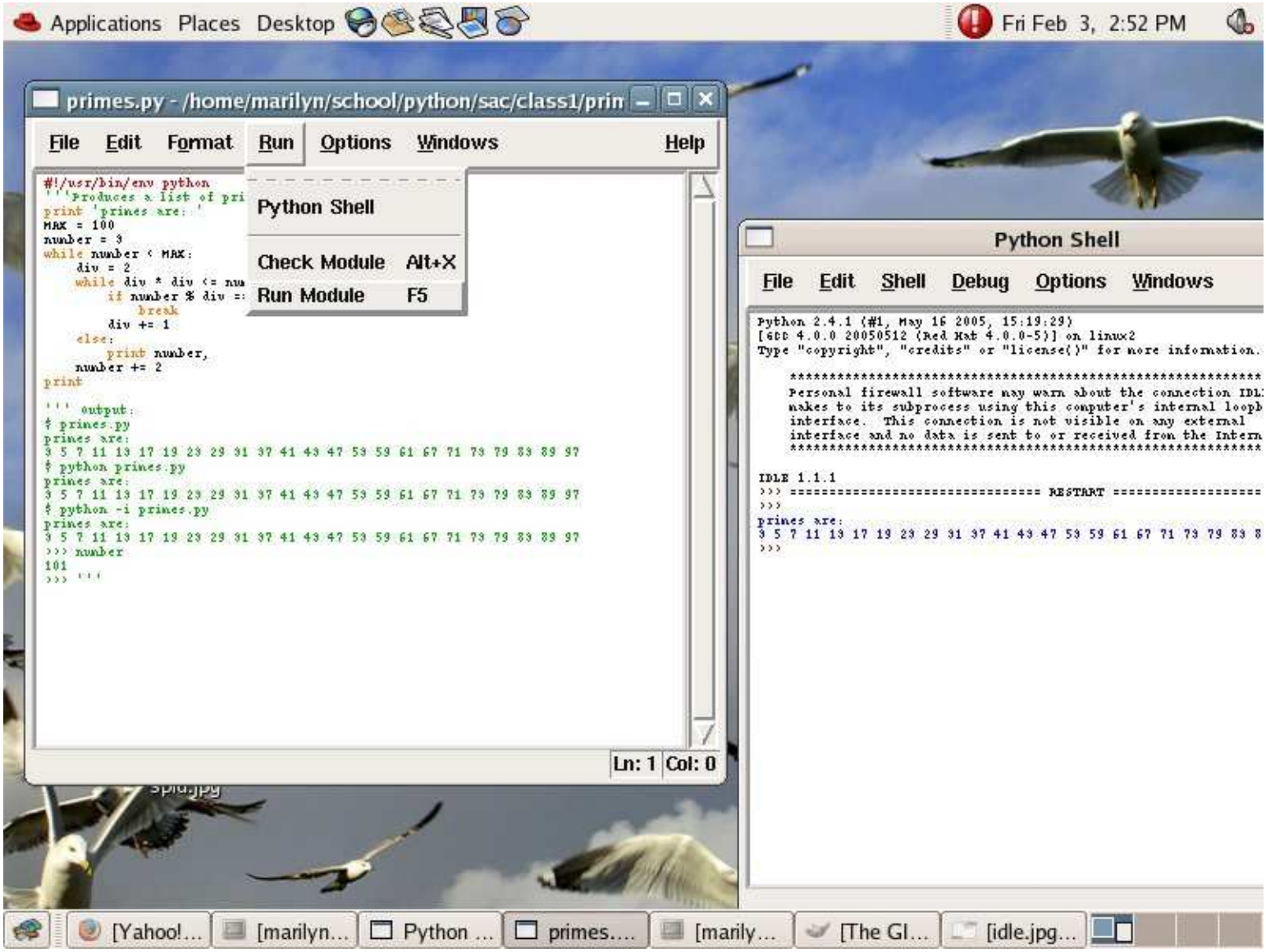


Figure 1: Idle Development Environment

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 Raw strings:
32 >>> print r"\n"
33 \n
34
35 These come in handy with regular expressions.
36 """
```

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.'
```

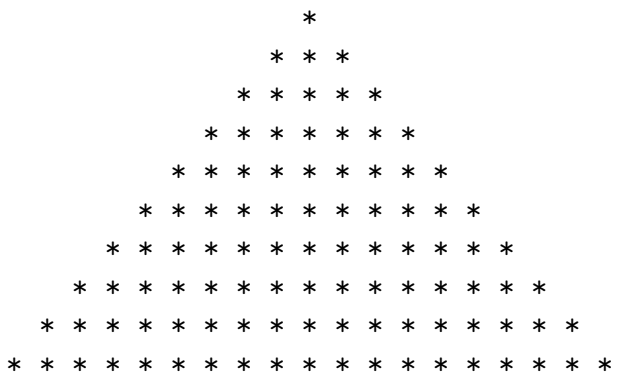
3. Write a program to produce this output — EXACTLY :

```
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