Acknowledgments

• Many of the slides I use were created by Dr. Chuck Severance, the author of our book.
  • You’ll typically recognize them by their black backgrounds.
  • They are *Copyright 2010–2014 Charles R. Severance* and are made available under a Creative Commons Attribution 4.0 License.

• Many of the slides I use were created by Dr. Jeremy Abramson of USC/ISI. He simply gave them to me to use in this class.

• Many of the slides I use were created by me.
How to Install Python 2

• **Not Python 3!**

• If you have a recent Mac, you’re done
  • Run the **Terminal** application
  • In the window that opens, type **Python**
  • (Read a bit about Unix commands, or ask)

• If you have a Windows PC, I have no idea
  • Search the web!

• Also, get a good simple text editor, preferably one that “understands” Python
  • For example, **TextWrangler** for the Mac

• Share information in class
Everything Is on the Internet

• Our book, with audio and video of lectures, slides
  • *Python for Informatics: Exploring Information*
• Search around pythonlearn.com for those things
• Other books, e. g., *Learn Python the Hard Way*
  • learnpythonthehardway.org/book/
• python.org
• stackoverflow.com
So Let’s Get Started

Unless there are questions?
Computers;
Python: Variables, Expressions, Statements
This, like many other slides are from Charles Severance’s “Python for Informatics.”
Definitions

- **Central Processing Unit**: Runs the Program - The CPU is always wondering “what to do next”? Not the brains exactly - very dumb but very very fast

- **Input Devices**: Keyboard, Mouse, Touch Screen

- **Output Devices**: Screen, Speakers, Printer, DVD Burner

- **Main Memory**: Fast small temporary storage - lost on reboot - aka RAM

- **Secondary Memory**: Slower large permanent storage - lasts until deleted - disk drive / memory stick
Software

Input and Output Devices

Central Processing Unit

Main Memory

Secondary Memory

What Next?

01001001
00111001

Machine Language
Programming in Python

• Programming used to be much more tedious

• Python is an *interpreted language*
  • You can type instructions and they’re executed on the spot
  • Or you can load a file with a *script*, and it will then be run

• Everything complicated that you’ll need has probably already be written: *Libraries*

• Pay attention to indentations — they matter!
More About Python

• Python was created by Guido van Rossum, while at Google

• It’s how we provide instructions to the CPU

• Python, like all computer languages, has a syntax

• If you don’t obey it, you will get a syntax error
  • Don’t despair

  • You can always try things out in your terminal window until you’re sure you understand how Python works

• Let’s try things
You Can Call Things Any Name…

- Except:

<table>
<thead>
<tr>
<th>and</th>
<th>as</th>
<th>assert</th>
<th>break</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>continue</td>
<td>def</td>
<td>del</td>
</tr>
<tr>
<td>elif</td>
<td>else</td>
<td>except</td>
<td>exec</td>
</tr>
<tr>
<td>finally</td>
<td>for</td>
<td>from</td>
<td>global</td>
</tr>
<tr>
<td>if</td>
<td>import</td>
<td>in</td>
<td>is</td>
</tr>
<tr>
<td>lambda</td>
<td>not</td>
<td>or</td>
<td>pass</td>
</tr>
<tr>
<td>print</td>
<td>raise</td>
<td>return</td>
<td>try</td>
</tr>
<tr>
<td>while</td>
<td>with</td>
<td>yield</td>
<td></td>
</tr>
</tbody>
</table>
Let’s Talk to Python...

```
$ python
[GCC 4.2.1 (Apple Inc. build 5646)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "hello world"
hello world
```
Go to Terminal/Python
### Sentences or Lines

<table>
<thead>
<tr>
<th>Variable</th>
<th>Operator</th>
<th>Constant</th>
<th>Reserved Word</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$=$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>$+$</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><code>print</code></td>
<td><code>x</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Python Scripts

• Interactive Python is good for experiments and programs of 3–4 lines long.

• Most programs are much longer, so we type them into a file and tell Python to run the commands in the file.

• In a sense, we are “giving Python a script”.

• As a convention, we add “.py” as the suffix on the end of these files to indicate they contain Python.
Interactive versus Script

• Interactive
  - You type directly to Python one line at a time and it responds

• Script
  - You enter a sequence of statements (lines) into a file using a text editor and tell Python to execute the statements in the file
Program Steps or Program Flow

• Like a recipe or installation instructions, a program is a sequence of steps to be done in order.

• Some steps are conditional - they may be skipped.

• Sometimes a step or group of steps are to be repeated.

• Sometimes we store a set of steps to be used over and over as needed several places throughout the program (Chapter 4).
Sequential Steps

When a program is running, it flows from one step to the next. As programmers, we set up “paths” for the program to follow.
Program:

```
x = 5
if x < 10:
    print 'Smaller'
if x > 20:
    print 'Bigger'
print 'Finis'
```

Output:

Smaller

Finis

print 'Finis'
**Repeated Steps**

Program:

```python
n = 5
while n > 0:
    print n
    n = n - 1
print 'Blastoff!'```

**Output:**

5
4
3
2
1
Blastoff!

Loops (repeated steps) have **iteration variables** that change each time through a loop. Often these iteration variables go through a sequence of numbers.
What Can Go Wrong?

- **Syntax errors**: You have violated the “grammar” rules of Python

- **Logic errors**: The program has good syntax, but there is a mistake in the order of the statements or perhaps a mistake in how the statements relate to one another

- **Semantic errors**: The program is perfectly correct but it does not do what you intended it to do
Python – Variables and Statements

- Variables
  - *Weakly* typed
    - Double-edged sword
  - Reserved words/keywords
  - Valid names

- Statements
  - Statements are where [small] things get done
  - Often also referred to as “lines” of code
Constants

- **Fixed values** such as numbers, letters, and strings are called “constants” because their value does not change.

- **Numeric constants** are as you expect.

- **String constants** use single quotes (') or double quotes (")

```python
>>> print 123
123
>>> print 98.6
98.6
>>> print 'Hello world'
Hello world
```
Variables

• A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”

• Programmers get to choose the names of the variables

• You can change the contents of a variable in a later statement

\[
\begin{align*}
x &= 12.2 \\
y &= 14
\end{align*}
\]
Variables

• A variable is a named place in the memory where a programmer can store data and later retrieve the data using the variable “name”

• Programmers get to choose the names of the variables

• You can change the contents of a variable in a later statement

\[
\begin{align*}
x &= 12.2 \\
y &= 14 \\
x &= 100
\end{align*}
\]
Python Variable Name Rules

1. Must start with a letter or underscore _

2. Must consist of letters and numbers and underscores

3. Case Sensitive
   - Good: spam eggs spam23 _speed
   - Bad: 23spam #sign var.12
   - Different: spam Spam SPAM
Sentences or Lines

\[ x = 2 \]  \hspace{2cm} \text{Assignment statement}

\[ x = x + 2 \]  \hspace{2cm} \text{Assignment with expression}

\text{print } x \hspace{2cm} \text{Print statement}

\begin{tabular}{cccc}
Variable & Operator & Constant & Reserved Word \\
\end{tabular}
Assignment Statements

- We assign a value to a variable using the assignment statement (=)

- An assignment statement consists of an expression on the right-hand side and a variable to store the result

\[ x = 3.9 \times x \times (1 - x) \]
## Arithmetic Operators

- Exponentiation looks different
  - $3 \times 2$ equals 9
- Asterisk is multiplication
- Modulus is the remainder after division:
  - $23 \% 6$ equals 5
  - “23 modulo 6”
  - $2 \% 5$ equals 2

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
</tr>
</tbody>
</table>
Order of Precedence

• **PEMDAS**
  • Parentheses
  • Exponentiation
  • Multiplication
  • Division
  • Addition
  • Subtraction
  • Left to right

• Which operator takes precedence?

  1 + 2 * 3 – 4 / 5 ** 6

• When in doubt, just use parentheses!
Python Numbers

• **1** is an *integer*

• **1.0** is a *floating point* number

• **Integer division is not what you expect!** Try these:

  ```
  print 10 / 2
  print 9 / 2
  print 9 / 10
  print 10.0 / 2.0
  print 9.0 / 10.0
  ```

• Using a decimal point in an operand changes the type of the resulting operation to *floating point*

• Try:  ```
  print 1 + 2 * 3 / 4.0 - 5
  ```  (What if I use 1.0 and 4?)

• Remember that Python is *weakly typed*?
What does “Type” Mean?

• In Python variables, literals and constants have a “type”

• Python knows the difference between an integer number and a string

• For example “+” means “addition” if something is a number and “concatenate” if something is a string

```python
>>> ddd = 1 + 4
>>> print ddd
5
>>> eee = 'hello ' + 'there'
>>> print eee
hello there
```

concatenate = put together
Type Matters

• Python knows what “type” everything is
• Some operations are prohibited
• You cannot “add 1” to a string
• We can ask Python what type something is by using the `type()` function

```python
>>> eee = 'hello ' + 'there'
>>> eee = eee + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: cannot concatenate 'str' and 'int' objects
```

```python
>>> type(eee)
<type 'str'>
>>> type('hello')
<type 'str'>
>>> type(1)
<type 'int'>
>>>```
Type Conversions

• When you put an integer and floating point in an expression, the integer is implicitly converted to a float

• You can control this with the built-in functions int() and float()

```python
>>> print float(99) / 100
0.99
>>> i = 42
>>> type(i)
<type 'int'>
>>> f = float(i)
>>> print f
42.0
>>> type(f)
<type 'float'>
>>> print 1 + 2 * float(3) / 4 - 5
-2.5
>>> ```
String Conversions

- You can also use `int()` and `float()` to convert between strings and integers.

- You will get an error if the string does not contain numeric characters.

```python
>>> sval = '123'
>>> type(sval)
<type 'str'>
>>> print sval + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int'
>>> ival = int(sval)
>>> type(ival)
<type 'int'>
>>> print ival + 1
124
>>> nsv = 'hello bob'
>>> niv = int(nsv)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```
String Operations

• Some operators apply to strings
  > + implies “concatenation”
  > * implies “multiple concatenation”

• Python knows when it is dealing with a string or a number and behaves appropriately

>>> print 'abc' + '123'
abc123
>>> print 'Hi' * 5
HiHiHiHiHi
>>>
User Input

• We can instruct Python to pause and read data from the user using the `raw_input()` function.

• The `raw_input('prompt')` function returns a string, after prompting the user with `prompt`.

• E. g.,

  ```python
  >>> x = raw_input('Who are you?')
  Who are you?
  ```
Converting User Input

• If we want to read a number from the user, we must convert it from a string to a number using a type conversion function

• (Later we will deal with bad input data)
Comments in Python

- Anything after a # is ignored by Python
- Why comment?
  - Describe what is going to happen in a sequence of code
  - Document who wrote the code or other ancillary information
  - Turn off a line of code – perhaps temporarily
Comments

- Use comments
- No really. Use comments
- “But what about using them in a string?”
- Make them meaningful

```
Vel = 5 # set Vel to 5  bad comment
Vel = 5 # Velocity in meters/second
```
# Get the name of the file and open it
name = raw_input('Enter file:')
handle = open(name, 'r')
text = handle.read()
words = text.split()

# Count word frequency
counts = dict()
for word in words:
    counts[word] = counts.get(word, 0) + 1

# Find the most common word
bigcount = None
bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count

# All done
print bigword, bigcount
Mnemonic Variable Names

```python
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print x1q3p9afd
```

What is this bit of code doing?
Mnemonic Variable Names

```python
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print x1q3p9afd

a = 35.0
b = 12.50
c = a * b
print c
```

What are these bits of code doing?
Mnemonic Variable Names

What are these bits of code doing?

```python
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print x1q3p9afd
```

```python
hours = 35.0
rate = 12.50
pay = hours * rate
print pay
```

```python
a = 35.0
b = 12.50
c = a * b
print c
```