• Learning to Program in the Python Programming Language
• IDLE development environment
• Python Shell Window & IDLE File Editor
• Running chaos.py
• Submitting a properly documented program
Using the Python Shell Window

The Python Shell Window provides access to Python's interactive mode.

You type in a Python statement at the >>> prompt and then hit the Enter key to send it to the Python interpreter.

If you ever find yourself in a situation where you seem to be hung and cannot get a new prompt, the interpreter is likely is a state where it is waiting for you to enter something specific (in its parse state). Hitting Ctrl-c will send a keyboard interrupt and should get you back to a prompt. It can also be used to interrupt any running command.

Colorization: Your code is colorized as you type it in based on Python syntax types.
Do these Tasks:
1. Start Python and IDLE

2. In the shell window (at the interpreter prompt >>>) try these statements:

```python
>>> print "hello, world"
>>> print 2+3
>>> 2+3
>>> print 9/5

>>> "hello, world"
>>> 99999999*99999999*99999999

More examples of printing strings.

>>> name = "John"
>>> print name
>>> longname = name + "Cleese"
>>> print longname
```
Indentation in Python

Code blocks are determined by indentation in Python.

If you type in the factorial function above you will find that when you hit Enter after the def or while clauses, the next line is automatically indented for you (as you enter a new block). When within a block, you will be automatically placed at the same level as the previous statement (above, this happens for the num = num - 1 statement). Finally, each press of the Backspace key will back you out of a level of indentation (above, it needs to be used before the return statement).

This is all you will normally be concerned about regarding indentation when in this Shell Window. It is also possible to change the indent level (which defaults here to an eight space tab), as well as indent or dedent (un-indent) a selected region. These commands are described later under the Edit a File.

**TASK 3. Enter the following simple function into the shell window, then try to have Python greet(yourname).**

```python
def greet(person):
    print "Hello", person
    print "How are you today?"
```
Word Completion

Typing an Alt-/ calls a word completion mechanism.

In the above example it is not necessary on the last line to enter in the full word divisible. We are able to just type d followed by Alt-/ and it is completed automatically.

Completion is based on those words that have been previously entered into the buffer. The first response will be the most recently entered word which matches. Successively entering Alt-/ will toggle through any other possibilities. So, in the above example entering an n followed by Alt-/ three times will toggle through num, not and then numList. If we started with nu this would be limited to num and numList.

TASK 4: Try this example of the divisible function.
Another nice feature that is worth using (because of the amount of typing it can save you) is the command history mechanism.

Once you've selected a previous command and brought it to the bottom, either by toggling through them (using Alt-p and Alt-n) or directly selecting it, you can edit it as desired before hitting the Enter key to have it sent to the interpreter.

**TASK 5:** Try at least 3 examples of reusing and editing commands, as above.
Function/Method Call Tips

As soon as you type in the opening bracket for a function or method call a small box will pop up below the current line giving you a "tip" with regards to the arguments that are expected.
Using the File Editor

IDLE's File Editor lets you create new files or browse and/or edit existing Python source files.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Unix Key(s)</th>
<th>Windows Key</th>
<th>Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open new window</td>
<td>Ctrl-x Ctrl-n</td>
<td>Ctrl-n</td>
<td>File:New</td>
</tr>
<tr>
<td>Open existing file</td>
<td>Ctrl-x Ctrl-f</td>
<td>Ctrl-o</td>
<td>File:Open...</td>
</tr>
<tr>
<td>Open module</td>
<td>Ctrl-x Ctrl-m</td>
<td>Ctrl-m</td>
<td>File:Open Module...</td>
</tr>
</tbody>
</table>

Open a File
New command will open an empty window (i.e. for when you want to start creating a Python source file from scratch). Open lets you navigate to and open any file on your hard drive.

Later we will use Open Module to use module files (i.e. shelve or shelve.py) that is on your module search path (sys.path) and it will find and open it for you.

In all cases, a new File Editor window will be opened with the name of the file and the path to it as its title (or called Untitled if its a new unsaved file).
Save a File

If the title of a buffer is delimited by *'s, this indicates it has changed since the last time it was saved to disk. There are three commands for saving a file.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Unix Key(s)</th>
<th>Windows Key</th>
<th>Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save</td>
<td>Ctrl-x Ctrl-s</td>
<td>Ctrl-s</td>
<td>File:Save</td>
</tr>
<tr>
<td>Save as</td>
<td>Ctrl-x Ctrl-w</td>
<td>Alt-s</td>
<td>File:Save As...</td>
</tr>
<tr>
<td>Save Copy As</td>
<td>Ctrl-x w</td>
<td>Alt-Shift-s</td>
<td>File:Save Copy</td>
</tr>
</tbody>
</table>

The first two are the standard "save to the existing name" and "save (possibly) to a new name". The third is the same as the second, except that it doesn't rename the buffer and tie it to the new file name.

Run a File

You can develop a module incrementally by first importing and then subsequently reloading it into the Shell Window to test it as you make changes. This can be done by using import or reload from within the Shell Window or by using the Run module command from the File Editor itself (Ctrl-F5).
Pasting Text and Edit a File

You can type or paste text into this window and the code you enter will be colorized as it is in the Shell Window.

**TASK 6:** Create new editing windows and paste each of the following loop statements; run these examples by selecting option ‘run module’ (Ctrl-F5):

```python
for i in range(10):
    print i

for j in range(1,10):
    print j

for h in range(10,20):
    print h
```

**TASK 7:** Try this example that accumulates a sum:

```python
sum = 0
for i in range(1,100):
    sum = sum + i
sum
```
## Moving around the Buffer

You can move around the buffer using the following key bindings. These commands work both in Shell Window and the File Editor.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Unix Key(s)</th>
<th>Windows Key(s)</th>
<th>Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move one space left</td>
<td>left arrow key, Ctrl-b</td>
<td>left arrow key, Ctrl-b</td>
<td>-</td>
</tr>
<tr>
<td>Move one space right</td>
<td>right arrow key</td>
<td>right arrow key</td>
<td>-</td>
</tr>
<tr>
<td>Move to start of current line</td>
<td>Home key, Ctrl-a</td>
<td>Home key, Ctrl-a</td>
<td>-</td>
</tr>
<tr>
<td>Move to the end of current line</td>
<td>End key, Ctrl-e</td>
<td>End key, Ctrl-e</td>
<td>-</td>
</tr>
<tr>
<td>Move up one line</td>
<td>up arrow key, Ctrl-p</td>
<td>up arrow key, Ctrl-p</td>
<td>-</td>
</tr>
<tr>
<td>Move down one line</td>
<td>down arrow key</td>
<td>down arrow key</td>
<td>-</td>
</tr>
<tr>
<td>Move up one page</td>
<td>Page Up key</td>
<td>Page Up key</td>
<td>-</td>
</tr>
<tr>
<td>Move down one page</td>
<td>Page Down key</td>
<td>Page Down key</td>
<td>-</td>
</tr>
<tr>
<td>Move to the top of the buffer</td>
<td>Ctrl-Home key</td>
<td>Ctrl-Home key</td>
<td>-</td>
</tr>
<tr>
<td>Move to the end of the buffer</td>
<td>Ctrl-End key</td>
<td>Ctrl-End key</td>
<td>-</td>
</tr>
<tr>
<td>Center window on selection</td>
<td>Ctrl-l</td>
<td>Ctrl-l</td>
<td>-</td>
</tr>
<tr>
<td>Find given entry</td>
<td>Ctrl-u Ctrl-u Ctrl-s</td>
<td>Ctrl-f</td>
<td>Edit:Find...</td>
</tr>
<tr>
<td>Find previous entry again</td>
<td>Ctrl-u Ctrl-s</td>
<td>Ctrl-g, F3</td>
<td>Edit:Find again</td>
</tr>
<tr>
<td>Find selection</td>
<td>Ctrl-s</td>
<td>Ctrl-F3</td>
<td>Edit:Find selection</td>
</tr>
<tr>
<td>Go to given line</td>
<td>Alt-g, Esc-g</td>
<td>Alt-g</td>
<td>Edit:Go to line</td>
</tr>
</tbody>
</table>
# Basic Editing

The following table contains key bindings that can be used for basic editing in the buffer.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Unix Key(s)</th>
<th>Windows Key(s)</th>
<th>Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delete character left of cursor</td>
<td>Backspace key</td>
<td>Backspace key</td>
<td></td>
</tr>
<tr>
<td>Delete character right of cursor</td>
<td>Delete key, Ctrl-d</td>
<td>Delete key, Ctrl-d</td>
<td></td>
</tr>
<tr>
<td>Cut selection</td>
<td>Ctrl-w</td>
<td>Ctrl-x</td>
<td>Edit:Cut</td>
</tr>
<tr>
<td>Paste into the buffer</td>
<td>Ctrl-y</td>
<td>Ctrl-v</td>
<td>Edit:Paste</td>
</tr>
<tr>
<td>Copy selection</td>
<td>Alt-w, Esc-w</td>
<td>Ctrl-c</td>
<td>Edit:Copy</td>
</tr>
<tr>
<td>Select whole buffer</td>
<td>Alt-a, Esc-a</td>
<td>Alt-a</td>
<td>Edit:Select All</td>
</tr>
<tr>
<td>Replace selection</td>
<td>Ctrl-r</td>
<td>Ctrl-h</td>
<td>Edit:Replace...</td>
</tr>
<tr>
<td>Expand (complete) word</td>
<td>Alt-/</td>
<td>Alt-/</td>
<td>Edit:Expand word</td>
</tr>
<tr>
<td>Previous command in history</td>
<td>Esc-p, Alt-p</td>
<td>Alt-p</td>
<td></td>
</tr>
<tr>
<td>Next command in history</td>
<td>Esc-n, Alt-n</td>
<td>Alt-n</td>
<td></td>
</tr>
<tr>
<td>Undo last change</td>
<td>Ctrl-z</td>
<td>Ctrl-z</td>
<td>Edit:Undo</td>
</tr>
<tr>
<td>Redo last change</td>
<td>Alt-z, Esc-z</td>
<td>Ctrl-y</td>
<td>Edit:Redo</td>
</tr>
</tbody>
</table>
DEALING with Python INDENTATION

- There are also a number of editing operations that work on selected regions of your file. The indent functions are obviously useful when adding or removing nested constructs from your code. The Tabify region function converts leading spaces in a line to tabs (8 spaces to a tab). The Untabify region function converts all tabs in the region to the correct number of spaces. These along with the indent functions are useful for converting to a standard format source code that has been copied in from various sources such as documents and web pages.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Key</th>
<th>Menu</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indent region</td>
<td>Ctrl-[</td>
<td>Edit:Indent Region</td>
</tr>
<tr>
<td>Dedent (unindent)</td>
<td>Ctrl-[</td>
<td>Edit:Dedent region</td>
</tr>
<tr>
<td>Comment out region</td>
<td>Alt-3</td>
<td>Edit:Comment out region</td>
</tr>
<tr>
<td>Uncomment region</td>
<td>Alt-4</td>
<td>Edit:Uncomment region</td>
</tr>
<tr>
<td>Tabify region</td>
<td>Alt-5</td>
<td>Edit:Tabify region</td>
</tr>
<tr>
<td>Untabify region</td>
<td>Alt-6</td>
<td>Edit:Untabify region</td>
</tr>
<tr>
<td>Format paragraph</td>
<td>Alt-q</td>
<td>Edit:Format Paragraph</td>
</tr>
</tbody>
</table>
Programming Simple Chaos

As discussed in lecture (see also Chapter 3 of James Gleick's book on chaos (Chaos - Making a New Science, 1987) scientists have discovered how very simple equations can exhibit chaotic behavior when particular parameters are manipulated. The example function we considered can be written in Python as follows:

```python
#File: chaos.py
# A simple program illustrating dynamic behavior of fish population
def main():
    print "This program illustrates a chaotic function"
    r = 3.9
    x = 0.1
    for i in range(20):
        x = r*x*(1-x)
        print x
    print "end of output"
main()
```

**TASK 8:** Paste this code into a new window and run the program.
What does this program do?

This program runs 20 iterations where the (next) population $x$ equals the rate of population growth $r$ multiplied by the current population $x$ multiplied by a limiting term $(1-x)$ that keeps the population within bounds (e.g. as fish population increases, food and oxygen becomes increasingly scarce and some of our fish die.)

**Task 9: Answer the following questions:**

a. Does the output ever reach an equilibrium when you increase the number of iterations (change the range(num_iterations))? 

b. Find a value of $r$ in which a single equilibrium point is reached?

c. Find a value of $r$ in which an equilibrium point is reached in which the population oscillates between 2 values?

d. Find a value of $r$ in which an equilibrium point is reached in which the population oscillates between 4 values?
**Properly Documenting your Python Programs**

**Task 10:** Every program that you submit must be properly documented using the style rules for the course. As an example of this, you will see below is a simple riddle program example that you can use as a template. Notice the comments and documentation. Modify the comments at the head of your program, and substitute your own riddle.

**Submission process:** You are to upload your program to the Blackboard system using the Digital Dropbox facility.
print "Welcome. Here is a riddle:\n"
print "If you say my name, then I am gone. Who am I?\n"
print "Do you need another clue? (y/n)"
response = raw_input()
if (response == 'y'):
  print "OK. Here is the next clue.\n"
  print "If you say nothing, then I am golden.\n"
  print "Hit enter to see answer"
  response = raw_input()
print "If you said \"silence\", then you are correct.\n"