

AICT Computing Series Winter 2011 Workshop

Practical Python Programming

charlene.nielsen@ualberta.ca
Thursday, February 24th, 2011
9:00 a.m. to 12:00 noon
ETLC 5-013



Practical Python Programming ETLC 5-013 Computer Access





1. Click the **middle button** to choose session number **5** "WINDOWS XP"
2. Click the CHANGE SESSION button
3. Log-in with your CCID and password (and click ACCEPT to the warning)
4. To access the internet: AUTHENTICATE using your CCID and password





Practical Python Programming

Windows XP




1. Right-click the **START** button to choose EXPLORE
2. Navigate to the TEMP directory on the C drive
3. In a web browser, download the following to C:\TEMP:
<http://tinyurl.com/ppp-zip>
4. Extract all/Unzip **ppp.zip** in to C:\Temp\ppp
5. Open **ppp1_tasks_w2011.pdf**, and then minimize for future use



Outline

- Who, when, what, where, why, how?
- Easy environments
- Programming primer
- Beverage break ~ 10:30 a.m.
- Coding conventions
- Fussing with the flow
- Excellent e-resources





Who am I?

Charlene Nielsen

ccn@ualberta.ca

GIS Analyst in Biological Sciences

<http://www.biology.ualberta.ca/facilities/gis>



Who is this workshop for?

- You, if you:
 - Know how to use a computer and the web
 - Have never programmed before but have a **strong desire to learn** how
 - Are willing to **type a lot** to make the information stick (no copy paste allowed)
 - Are an experienced programmer and want to **play** with a new language

Who uses it?

- Web Development
 - YAHOO! Google
 - ZOPE Corporation
 - LWN.net
- Graphics
 - INDUSTRIAL LIGHT & MAGIC
 - blender™
- Financial
 - ALTIS INVESTMENT MANAGEMENT
 - BELLCO CREDIT UNION
- Science
 - NASA National Aeronautics and Space Administration
 - NRC-CNRC
 - NOAA
 - esri

<http://wiki.python.org/moin/OrganizationsUsingPython>

Who uses it?

- Electronic Design Automation
 - CIRANOVA analog layout automation
- Software Development
 - redhat
 - sgi
 - NOKIA Connecting People
- Education
 - UNIVERSITY REGISTRAR UNIVERSITY of CALIFORNIA - IRVINE
 - THE NEW ZEALAND DIGITAL LIBRARY The University of Waikato
- Business Software
 - AMS ADVANCED MANAGEMENT SOLUTIONS
 - thawte
 - IBM
- Government
 - CENTRAL INTELLIGENCE AGENCY

<http://wiki.python.org/moin/OrganizationsUsingPython>

Who is responsible for Python?

- Created by Guido van Rossum
 - born 31 Jan 1956 (Netherlands)
 - currently works at Google (since 2005)



- Actively maintained by worldwide users (including van Rossum)

When?

- 1989 "hobby" programming project
 - interpreter for a new scripting language descendant of ABC that would appeal to Unix/C hackers
 - needed a name that was short, unique, and slightly mysterious – so was named Python as a working title
- 1991 code first published (version 0.9.0)
- 2001 the Python Software Foundation was formed

More history:

<http://docs.python.org/release/2.6.6/license.html>
http://en.wikipedia.org/wiki/History_of_Python



When to (not) use it?


- **PRO:** Suitable for everyday tasks, therefore minimizing the amount of time a programmer spends on a project
- **CON:** Often less efficient than languages such as Java, C, and C++, therefore, not good when creating a new operating system



What is it?


- A remarkably **powerful**, dynamic, object-oriented, interpreted, high-level programming language that is used in a wide variety of application domains
- Named after the comedy troupe: **Monty Python**






What release: 2 or 3?

- “Python 2.x is the status quo, Python 3.x is the shiny new thing.”
- 2.7 is the last major release for 2.x, with statement of extended support
- 3.1 is the intended future of the “cleaned up” language, with less regard for backwards compatibility
- A lot of third party software doesn't work on 3.x yet...
- “Well written 2.x code will actually be a lot like 3.x code”
- Handy tools for compatibility:
 - <http://wiki.python.org/moin/2to3>
 - <http://wiki.python.org/moin/3to2>




I'm the tried and true blue #2!



I'm the clean and shiny #3!

<http://wiki.python.org/moin/Python2orPython3>



What can it do?

Run Web sites

And more!

Build test suites for C or Java code

Write GUI interfaces

Process large XML data sets

Control number-crunching code on supercomputers

Make a commercial application scriptable by embedding the Python interpreter inside it

Image from "Head First Python" book by Paul Berry

<http://www.headfirstlabs.com/books/hfpython/>

Where can it be found?

- www.python.org
- www.activestate.com
- Automatically with certain operating systems (e.g. Mac and Linux)
- Installed alongside software applications that incorporate its scripting functionality (e.g. ESRI ArcGIS)

*Follow the
easy online
installation
instructions!*



Where can it be used?

- All major operating systems
 - Windows
 - Linux/Unix
 - OS/2
 - Mac
 - Amiga



Windows



ubuntu®



– OS/2



– Mac

Mac



– Amiga

And operating systems on mobile devices!



Why use it?

- **Simplicity and ease of understanding**
 - Programs look neat and clean
 - Has few unnecessary symbols
 - Straightforward English names instead of the cryptic syntax common in other languages

Note: 'import antigravity' only works in version 3.x, print statement only in 2.x
<http://xkcd.com/353/>

Why use it?

- **Programmability**
 - Wide range of readymade libraries that can be (freely!) used in your own programs; i.e. “batteries included”
 - Supports, but doesn't force, object-oriented programming (OOP)
 - Integration with other languages (e.g. Java, C)

"Python Foot" graphic by [David Day](#)




Why use it?



- Free
 - Doesn't cost – freely usable and distributable – even for commercial use
 - Open source code
 - Sizable community of developers with online support groups



<http://www.pycon.org>



How can it be used?

- Scripting
 - 'gluing' together commands for other software applications
- Programming
 - text processing, web site development, email, scientific computing
 - <http://www.devsource.com/c/a/Languages/More-Than-Five-Things-You-Didn't-Know-You-Could-Do-With-Python/>
 - Python does COM
 - Python does .NET
 - Python does Java better than Java
 - Python is high-performance
 - Python talks to hardware
- Education
 - MIT and other institutions use Python as the programming language in their introductory computer science courses

How can it be learned?


- Official Python tutorial
<http://docs.python.org/release/2.6.6/tutorial/index.html>
- Online resources, including “free” e-books through UofA’s Libraries (listed at the end)
- Hands-on now!



Easy environments



- Command line
 - The interpreter
- IDE (IDLE):
Integrated
DeveLopment
Environment
 - Interactive window
 - Script window
- Other software
(not used in this workshop)




The Python Interpreter

We're going to skip the command line, and go on to...



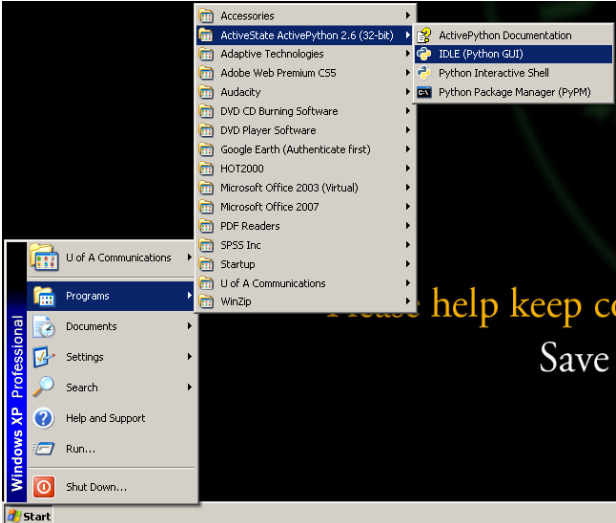
```

Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print "And now for something completely different..."
And now for something completely different...
>>>
    
```



IDLE for Python in ETLC 5-013

Click START >>> PROGRAMS
 >>> ACTIVESTATE ACTIVEPYTHON 2.6 (32-bit) >>> IDLE (PYTHON GUI)





IDLE: Interactive window

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.6.5 (r265:79096, Mar 19 2010, 21:48:26) [MSC v.1500 32 bit
(Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 2.6.5
>>> print "And now for something completely different..."
And now for something completely different...
>>> |
```


<http://docs.python.org/library/idle.html>
http://hkn.eecs.berkeley.edu/~dyoo/python/idle_intro/index.html



Interactive window menu: File

<p>Create a new editing window</p> <p>Open an existing file</p> <p>Open a list of recent files</p> <p>Save current window to the associated file (unsaved windows have a * before and after the window title)</p> <p>Save current window to different file without changing the associated file</p> <p>Close all windows, quit (asks to save if unsaved)</p>		<p>Open an existing module (searches sys.path)</p> <p>Show classes and methods in current file</p> <p>Show sys.path directories, modules, classes and methods</p> <p>Save current window to new file, which becomes the associated file</p> <p>Print the current window</p> <p>Close current window (asks to save if unsaved)</p>
--	--	---

Interactive window menu: Edit




	Edit	Shell	Debug	Options	Windows	Help
Undo last change to current window (A maximum of 1000 changes may be undone)	U <u>ndo</u>					Ctrl+Z
	R <u>edo</u>					Ctrl+Shift+Z
Copy a selection into system wide clipboard, then delete the selection	C <u>u</u> t					Ctrl+X
	C <u>o</u> py					Ctrl+C
	P <u>a</u> ste					Ctrl+V
Insert system wide clipboard into window	S <u>e</u> lect <u>A</u> ll					Ctrl+A
Ask for a line number and show that line	F <u>i</u> nd...					Ctrl+F
	F <u>i</u> nd <u>A</u> gain					Ctrl+G
	F <u>i</u> nd <u>S</u> election					Ctrl+F3
	F <u>i</u> nd in <u>F</u> iles...					Alt+F3
	R <u>e</u> place...					Ctrl+H
	G <u>o</u> to <u>L</u> ine					Alt+G
	S <u>h</u> ow call tip					Ctrl+backslash
	S <u>h</u> ow <u>C</u> ompletions					Ctrl+space
	S <u>h</u> ow surrounding <u>p</u> arens					Ctrl+0
	E <u>x</u> pand <u>W</u> ord					Alt+/

more...


Interactive window menu: Edit

continued



	Edit	Shell	Debug	Options	Windows	Help
	U <u>ndo</u>					Ctrl+Z
	R <u>edo</u>					Ctrl+Shift+Z
	C <u>u</u> t					Ctrl+X
	C <u>o</u> py					Ctrl+C
	P <u>a</u> ste					Ctrl+V
	S <u>e</u> lect <u>A</u> ll					Ctrl+A
Search for the string in the selection	F <u>i</u> nd...					Ctrl+F
Open a search dialog box for searching files	F <u>i</u> nd <u>A</u> gain					Ctrl+G
	F <u>i</u> nd <u>S</u> election					Ctrl+F3
Open a small window with function param hints	F <u>i</u> nd in <u>F</u> iles...					Alt+F3
	R <u>e</u> place...					Ctrl+H
	G <u>o</u> to <u>L</u> ine					Alt+G
	S <u>h</u> ow call tip					Ctrl+backslash
	S <u>h</u> ow <u>C</u> ompletions					Ctrl+space
	S <u>h</u> ow surrounding <u>p</u> arens					Ctrl+0
Highlight the surrounding parenthesis	E <u>x</u> pand <u>W</u> ord					Alt+/

Interactive window menu: Other



Shell Debug Options Windows

- View Last Restart: F6
- Restart Shell: Ctrl+F6

Look around the insert point for a filename and linenummer, open the file, and show the line

Show the stack traceback of the last exception

Toggles the window between configured size and maximum height.

The rest of this menu lists the names of all open windows; select one to bring it to the foreground (deiconifying it if necessary).

Shell and Debug menus are not available in the script window

Scroll the shell window to the last restart

Restart the interpreter with a fresh environment

Run commands in the shell under the debugger

Open stack viewer on traceback

Zoom Height: Alt+2

Python Shell


Debug Options Windows H

- Go to File/Line
- Debugger
- Stack Viewer
- Auto-open Stack Viewer

Windows Help

- Zoom Height: Alt+2
- *Python Shell*

Interactive window menu: GTK*



Options Windows Help

- Configure IDLE...

Version, copyright, license, credits

Access local Python documentation, if installed. Otherwise, access www.python.org.

(Additional Help Sources may be added here using Options > Configure IDLE...)

*GTK = Good To Know!

Open a configuration dialog. Fonts, indentation, keybindings, and color themes may be altered. Startup Preferences may be set, and Additional Help Sources can be specified. On MacOS X this menu is not present, use menu 'IDLE > Preferences...'

About IDLE

IDLE Help

Python Docs: F1

Display summary file on all menu items

Options Windows Help

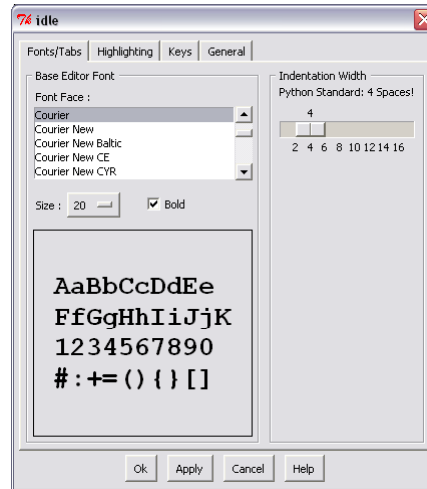
- Configure IDLE...

Help

- About IDLE
- IDLE Help
- Python Docs: F1

Now try this...

- Click Help > IDLE Help
 - Close when finished reading
- Click Options > Configure IDLE
 - Examine each tab and modify as needed
 - E.g. change font (Fonts/Tabs tab)
 - E.g. add custom web/file help link/path (General tab and click Add button)



Programming primer

- Program
 - A set of instructions or recipe that tells the computer what you want it to do





Programming primer

- A Python program is divided into **lines**
 - Each line physically ends before a newline
 - Lines that are **blank** (contain only spaces, tabs, formfeeds and possibly a comment) are **ignored**
 - Breaking up statements on separate lines will break your program!

```
thesunshines = True
```



Programming primer

- Indentation and whitespace required
 - Leading whitespace (i.e. spaces) at the beginning of a logical line is used to compute the indentation level of the line, which in turn is used to determine the **grouping** of statements
 - Trailing whitespace is ignored
 - All lines you want grouped together must be indented identically!



```
thesunshines = True
while thesunshines:
    print "Skip work and go to the beach"
thesunshines = False
```



Programming primer

- Comments

- Helps explain what the code is doing
- Ignored when executed
- Two ways to indicate comments:

```
# type comment here
```

```
""" type comment here """
```

- Command prompts (interactive mode only)

```
>>>
```

```
...
```



Now try this...

- Type the following in the interactive window:

```
>>> import this
```

```
>>> help()
```

```
>>> import
```

```
>>> topics
```

```
>>> NUMBERS
```

```
>>> strings          # note the lower case
```

Do NOT type the >>>

- Type quit or press Ctrl-C to exit the help system



Now try this...

- Type the following in the interactive window:

```
>>> help(keywords)           # now retype with quotes around 'keywords'
>>> help('print')
>>> print 'what is a blue moon?'
>>> abluemoon = 'second to last full moon in a four-moon season'
>>> print abluemoon
>>> for once in abluemoon:
    print once                # this is looping, we'll learn about it later
```

- Type Alt-P a couple of times (and try Alt-N)

Do NOT type the >>>



Building blocks



- Difference between a statement and an expression:
 - Expression is something
 - Statement does something
(or, rather, tells the computer to do something; e.g. print or assignment)
- Expressions consist of:
 - Literal values
 - Variables
 - Operators
 - Parentheses (to override the built-in precedence system)
 - And some other things...

http://docs.python.org/reference/lexical_analysis.html



Literals

- Notations for constant values of some built-in types (i.e. the stuff you type directly)
 - **Numeric** = plain integers, long integers, floating point numbers, and imaginary numbers
 - **Strings** = text enclosed in 'single quotes' or "double quotes"

```
help('STRINGS')  
help('NUMBERS')
```



Variables

- A name that represents or refers to a value (i.e. the stuff you want stored for various uses)
 - **Dynamic** – no need to define the type up front
 - Created through an **assignment** statement that gives them values using the '=' operator
 - If you don't have an assignment (i.e. a name to the left of the '=') then Python stores it in the default result variable '_' (note: type '_' at the command prompt to view current value)

*"Programmers use these variable names to make their code read more like English, and because they have lousy memories. If they didn't use good names for things in their software, they'd get lost when they tried to read their code again."
(Learn Python the Hard Way)*



Operators

- Tokens that work with or operate on values
- Includes:
 - Arithmetic
 - Assignment
 - Comparison
 - Bitwise
 - Logical
 - Membership
 - Identity

(Note: The smaller-font ones are not included in the next set of slides; look them up in the help!)

`help('OPERATORS')`

http://www.tutorialspoint.com/python/python_basic_operators.htm



Operators: Arithmetic

+	Add	(Note: Concatenation when used with strings)
-	Subtract	
*	Multiply	(Note: Repetition when used with strings)
**	Exponential (power) calculation	
/	Divide	
%	Modulus – return remainder from division	
//	Floor Division – return quotient from division	

Operators: Comparison

==	Equal to*
!=	Not equal to (also <>)
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

*Remember that a single '=' sign is the assignment operator!

Operators: Assignment

=	Simple assignment operator* (assigns values from right side operands to left side operand)
+=	Add AND assignment operator (add right operand to the left operand and assign the result to left operand)
-=	Subtract AND assignment operator (subtract right operand from the left operand and assign the result to left operand)
*=	Multiply AND assignment operator (multiply right operand with the left operand and assign the result to left operand)
/=	Divide AND assignment operator (divide left operand with the right operand and assign the result to left operand)

*Remember that a double '==' sign is the equality operator!



Keywords

- A **reserved** word or identifier that has a particular meaning to the programming language
- They must be spelled **exactly** as typed here:

and	elif	if	print
as	else	import	raise
assert	except	in	return
break	exec	is	try
class	finally	lambda	while
continue	for	not	with
def	from	or	yield
del	global	pass	

help('keywords')

http://docs.python.org/reference/lexical_analysis.html#keywords



Now try this...

- Type the following in the interactive window:

```
>>> 1 + 2 + 3
```

```
>>> _
```

```
>>> _ - 4
```

```
>>> a = 1
```

```
>>> b, c = 2, 3
```

```
>>> A + b * c
```

then type a + b * c

```
>>> 'stuff'
```

```
>>> _
```

Case is important!
A is NOT the same as a

Do NOT type the >>>

Now try this...

- Type the following in the interactive window:

```
>>> # calculator expressions
```

```
>>> # commenting and docstrings
```

```
>>> # print statements
```

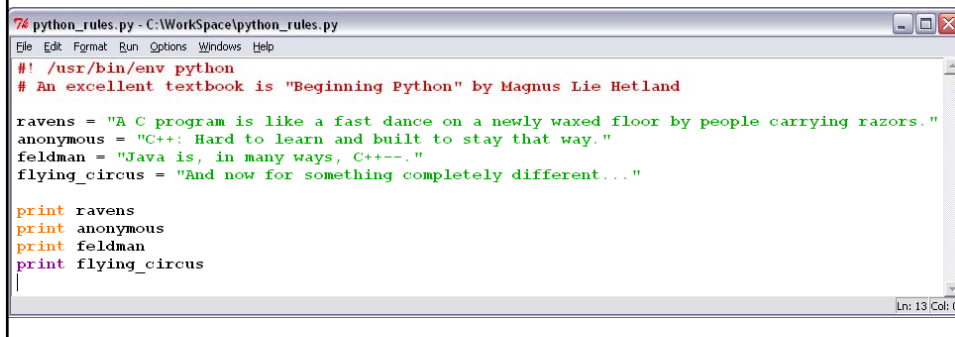
```
>>> # sequences
```

*Samples demonstrated
and provided in
the PDF document*

- IDLE is well-suited for testing out bits of code to see how it works!

IDLE: Script window

- From the menu of the interactive (Python Shell) window, click **File >>> New Window**
 - A new blank window (a.k.a. **file editor**) is used to type our programs in




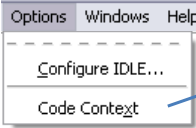
```
python_rules.py - C:\Workspace\python_rules.py
File Edit Format Run Options Windows Help
#!/usr/bin/env python
# An excellent textbook is "Beginning Python" by Magnus Lie Hetland

ravens = "A C program is like a fast dance on a newly waxed floor by people carrying razors."
anonymous = "C++: Hard to learn and built to stay that way."
feldman = "Java is, in many ways, C++-."
flying_circus = "And now for something completely different..."

print ravens
print anonymous
print feldman
print flying_circus
```


Script window menu





Options Windows Help

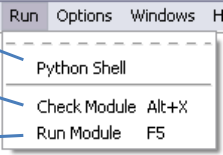
Code Context

Open a pane at the top of the edit window which shows the block context of the section of code which is scrolling off the top or the window

Open or wake up the Python shell window

Run a syntax check on the module

Execute the current file in the `__main__` namespace



Run Options Windows H

Python Shell


Check Module Alt+X

Run Module F5

Shell and Debug menus are not available in the script window
But, all the previously shown ones are: File, Edit, Windows, Help!

Run menu is not available in the [interactive window](#)

Script window menu: Format



Shift selected lines right 4 spaces

Shift selected lines left 4 spaces

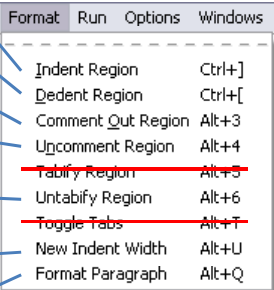
Insert ## in front of selected lines

Remove leading # or ## from selected lines

Turn *all* tabs into the right number of spaces

Open dialog to change indent width

Reformat the current blank-line-separated paragraph



Format Run Options Windows

Indent Region Ctrl+[

Dedent Region Ctrl+]

Comment Out Region Alt+3

Uncomment Region Alt+4

~~Tabify Region Alt+5~~

~~Untabify Region Alt+6~~

~~Toggle Tabs Alt+T~~

New Indent Width Alt+U

Format Paragraph Alt+Q

Format menu is not available in the [interactive window](#)



A script

- A text file containing the statements that comprise a Python program
- *Reusable, reusable, reusable* – once created, the script can be executed over and over without having to retype it each time
- Any file editor (that saves as plain ASCII) can be used, but name the file with a `.py` extension
- But don't forget about interactive mode completely, it makes an excellent testing ground



Coding conventions

- Style Guide for Python Code
<http://www.python.org/dev/peps/pep-0008/>
 - 4 spaces per indentation (no tabs ever)
 - Maximum line length of 79 characters
 - Always import modules after initial comments or docstrings and import them separately (i.e. don't scatter them throughout your code)
 - Name:
 - modules with lowercase
 - classes with CapWords
 - functions with lower_case_with_underscores
 - constants with UPPERCASE (and possibly “_”)
 - Use blank lines to separate groups of related code
 - Surround operators with single space on either side
 - Avoid extraneous whitespace in other situations
 - Consistency! You really should read through PEP 8



Now try this...

- Type some of the previous statements you've already tried in to the **script** window

*Samples demonstrated
and provided in
the PDF document*

- Save with a meaningful name and a **.py** extension to your working directory
- Optionally, click Run >>> Check Module
- Click Run >>> Run Module (or the F5 key)



Fussing with the flow

- Control the behaviour of a script by testing conditions and looping through values
- Flow control statements (similar to other programming languages) specify how the program is executed
 - **if**
 - **while**
 - **for**
 - Also **try** and **with** (but not covered in this workshop)
- Keyword statement (typically) followed by a colon and an indented block

http://docs.python.org/release/2.6.6/reference/compound_stmts.html



if

- The **if** statement is used for conditional execution, a.k.a. testing values and making decisions
- Details: lets you perform an action (another statement or more) if a given condition is true; if the condition is false, then the block is not executed
- Remember: when testing equality use `==` (and not `=`)
- Extending the testing:
 - **elif** allows you to check multiple expressions for truth value and execute a block of code as soon as the first one of the conditions evaluates to true (all following elif expressions are ignored)
 - **else** is a catch-all in case previous condition(s) not met

http://docs.python.org/release/2.6.6/reference/compound_stmts.html



if

- Generic syntax:

```
if expression:
    statement(s)
elif expression:
    statement(s)
else:
    statement(s)
```

← *Most basic minimum coding*
- Can be abbreviated as a conditional statement:
`x = true_value if condition else false_value`

http://docs.python.org/release/2.6.6/reference/compound_stmts.html



while

- The **while** statement is used for repeated execution as long as an expression is true
- Details: keeps executing (another statement or more) until the condition becomes false
- Remember: to code a finite loop; i.e. prevent the possibility that this condition never resolves to a false value (unless an infinite loop is actually desired)
- Generic syntax:

```
while expression:  
    statement(s)
```

http://docs.python.org/release/2.6.6/reference/compound_stmts.html



for

- The **for** statement is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object
- Details: causes a section of a program to be repeated a certain number of times by iterating (i.e. counting each)
- Remember: this can loop through list-like objects and/or numeric ranges
- Generic syntax:

```
for iterating_var in sequence:  
    statements(s)
```

http://docs.python.org/release/2.6.6/reference/compound_stmts.html



Now try this...

- Type the following in the **script** window:

```
>>> # if
>>> # X if C else Y
>>> # while
>>> # for
```

*Samples demonstrated
and provided in
the PDF document*



Built-in functions



- A series of statements which returns some value to a caller
(it can also be passed zero or more arguments for its use/execution)
- Objects that can be used by all Python code without the need of an import statement

```
dir(__builtins__)
```

<http://docs.python.org/release/2.6.6/library/functions.html>



Built-in functions

Some of the tools that are always available to use:

<code>abs(x)</code>	<code>globals()</code>
<code>bin(x)</code>	<code>help([object])</code>
<code>cmp(x, y)</code>	<code>hex(x)</code>
<code>complex([real[, imag]])</code>	<code>id(object)</code>
<code>dict([arg])</code>	<code>input([prompt])</code>
<code>dir([object])</code>	<code>int([x[, base]])</code>
<code>divmod(a, b)</code>	<code>isinstance(object, classinfo)</code>
<code>enumerate(sequence[, start=0])</code>	<code>iter(o[, sentinel])</code>
<code>eval(expression[, globals[, locals]])</code>	<code>len(s)</code>
<code>file(filename[, mode[, bufsize]])</code>	<code>list([iterable])</code>
<code>float(x)</code>	<code>locals()</code>
<code>format(value[, format_spec])</code>	<code>long([x[, base]])</code>

See <http://docs.python.org/release/2.6.6/library/functions.html> for complete listing



Built-in functions

More selected tools:

<code>map(function, iterable, ...)</code>	<code>set([iterable])</code>
<code>max(iterable[, args...][, key])</code>	<code>sorted(iterable[, cmp[, key[, reverse]])</code>
<code>min(iterable[, args...][, key])</code>	<code>str([object])</code>
<code>oct(x)</code>	<code>sum(iterable[, start])</code>
<code>open(filename[, mode[, bufsize]])</code>	<code>tuple([iterable])</code>
<code>pow(x, y[, z])</code>	<code>type(object)</code>
<code>range([start[, stop[, step]])</code>	<code>type(name, bases, dict)</code>
<code>raw_input([prompt])</code>	<code>vars([object])</code>
<code>reversed(seq)</code>	<code>xrange([start[, stop[, step]])</code>
<code>round(x[, n])</code>	<code>zip([iterable, ...])</code>

See <http://docs.python.org/release/2.6.6/library/functions.html> for complete listing



User-defined functions

- A block of organized, reusable code that is used to perform a single related action, that YOU create
- Similar to procedures, subroutines, and functions in other programming languages, but may or may not return a value
- Generic syntax:

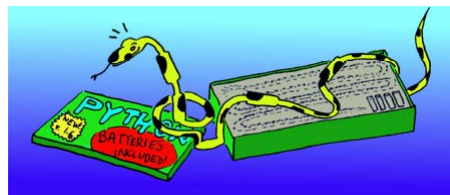
Required keyword: **def**
defines the function

```
def functionname( parameters ):  
    """function_docstring"""  
    function_suite  
    return [expression]
```



Modules

- Basically, subprograms that define things, e.g. functions, classes, variables
- Built-in (as opposed to user-defined) modules are also called **standard libraries**



<http://docs.python.org/release/2.6.6/library>
<http://www.doughellmann.com/PyMOTW/>



Modules

- Different ways to grab these extra tools:
 1. **import** <module name>
 - Requires module name as prefix to tools
e.g. `import random`
`randvalue = random.random() * 100`
 2. **from** <module name> **import** <function>
 - Reduces typing for commonly used tools
e.g. `from random import random`
`randvalue = random() * 100`
 3. **from** <module name> **import** *
(this way is not advised because it causes clutter)



The Cheese Shop

- The **Python Package Index** is one-stop 'shopping' for all registered module packages:

<http://pypi.python.org/pypi>

- Searchable!



Certainly, sir.
What would
you like?

Now then,
some cheese
please, my
good man.



Some final notes

- As with all things and in Python, it's
Easier to Ask Forgiveness
than Permission
- I.e. try and perform an operation; if all goes well,
great; if not, ask for forgiveness
- Although, not demonstrated in this workshop, this is
error handling and is done with **try/except**
- The FAQ is a worthwhile read:
<http://docs.python.org/faq/index.html>



Five tips for Python

1. Don't work too hard – let Python do the work for you by
using the stuff built in to its standard library
2. Don't write a loop when a comprehension will do –
built-in syntax for transforming one data structure (lists,
dictionaries) into another version of itself without resorting to
iteration
3. Learn as you go, not all at once, and learn constantly
(remember, concepts are the important things)
4. IDLE is often good for quick jobs (and has some useful
tools)
5. Don't reinvent the wheel – check the Python Package Index
(PyPI) first

<http://answers.oreilly.com/topic/2227-how-to-tackle-python-head-first/>


Skeleton script suggestion

- Set up a `generic_code.py` (name it what you want)
- Use comments/docstring at the top to indicate the filename, date(s), your name, optional contact info, and details about what the script requires and does
- Type out the basic structure of your common coding needs; including `import <commonly used module(s)>`
- Open the file when you're ready to start a new program, simply saving as a new appropriate name before modifying

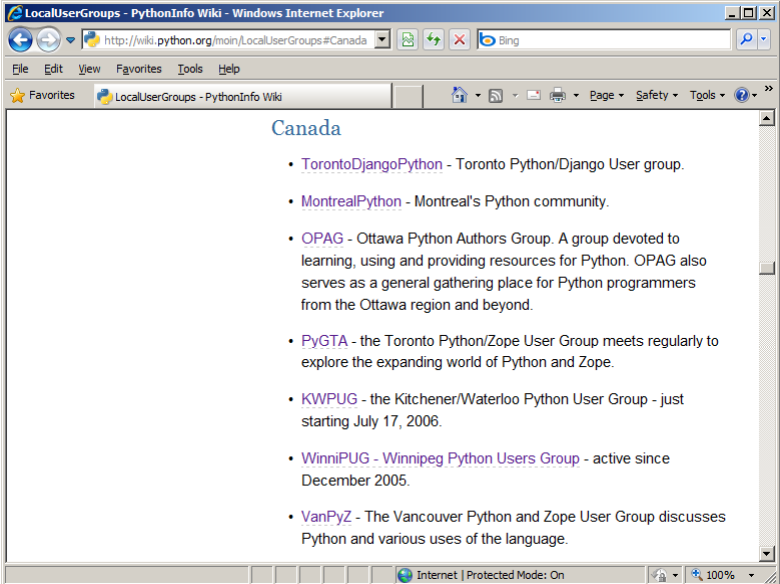


So, are you now a fledgling pythoner?
OR
Do you feel like you've found the 'holy grail'
of programming languages?





PyEd, anyone?



LocalUserGroups - PythonInfo Wiki - Windows Internet Explorer

http://wiki.python.org/moin/LocalUserGroups#Canada

Canada


- [TorontoDjangoPython](#) - Toronto Python/Django User group.
- [MontrealPython](#) - Montreal's Python community.
- [OPAG](#) - Ottawa Python Authors Group. A group devoted to learning, using and providing resources for Python. OPAG also serves as a general gathering place for Python programmers from the Ottawa region and beyond.
- [PyGTA](#) - the Toronto Python/Zope User Group meets regularly to explore the expanding world of Python and Zope.
- [KWPUG](#) - the Kitchener/Waterloo Python User Group - just starting July 17, 2006.
- [WinniPUG](#) - [Winnipeg Python Users Group](#) - active since December 2005.
- [VanPyZ](#) - The Vancouver Python and Zope User Group discusses Python and various uses of the language.



Future Python group activities

- More AICT Workshops with Python
- Software Carpentry:
<http://software-carpentry.org>
- MIT Intro to Computer Science Programming:
<http://academicearth.org/courses/introduction-to-computer-science-and-programming>
- Read the related article on the importance of programming skills:
<http://www.python.org/doc/essays/cp4e.html>



<http://www.python.org/community/sigs/current/edu-sig/>




E-resources

(online links to software, documentation, free books, tutorials, sample code, and articles)

- <http://www.python.org/>
- <http://hetland.org/writing/instant-python.html>
- <http://rgruet.free.fr/PQR26/PQR2.6.html>
- <http://diveintopython.org/>
- <http://www.headfirstlabs.com/books/hfpython/>
- <http://learnpythonthehardway.org/>
- <http://greenteapress.com/thinkpython/>
- http://homepage.mac.com/s_lott/books/python.html
- <http://www.tutorialspoint.com/python/>
- <http://www.developer.com/lang/other/article.php/3624681/Python-Tutorial-Index-Page.htm>
- <http://www.swaroopch.com/notes/Python>
- <http://www.korokithakis.net/tutorials/python>
- <http://oreilly.com/python/>
- http://en.wikipedia.org/wiki/Comparison_of_programming_languages
- <http://code.activestate.com/recipes/langs/python/>
- <http://pypi.python.org/pypi>
- <http://www.python.org/doc/humor/>



UofA e-books

(subscription is accessible through an on-campus computer with a UofA IP address)

<http://www.springerlink.com/>

- Pro Python, Marty Alchin
- Python Programming Fundamentals, Kent D. Lee
- A Primer on Scientific Programming with Python, Hans Petter Langtangen
- Python Scripting for Computational Science, Hans Petter Langtangen
- Beginning Python Visualization: Crafting Visual Transformation Scripts, Shai Vaingast
- Dive Into Python, Mark Pilgrim
- Foundations of Agile Python Development, Jeff Younker
- Beginning Python: From Novice to Professional, Magnus Lie Hetland

<http://site.ebrary.com/lib/ualberta/home.action>

- Python Programming for the Absolute Beginner, Michael Dawson
- Gray Hat Python: Python Programming for Hackers and Reverse Engineers, Justin Seitz
- Python: Create - Modify - Reuse, James O. Knowlton
- Python Power!: The Comprehensive Guide, Matt Telles

Most recently published (and latest editions) listed from top to bottom



UofA e-books

(subscription is accessible through an on-campus computer with a UofA IP address)

<http://proquest.safaribooksonline.com/>

- Programming Python, Mark Lutz
- Python Algorithms: Mastering Basic Algorithms in the Python Language, Magnus Lie Hetland
- Head First Python, Paul Barry
- Python Testing, Daniel Arbutckle
- The Quick Python Book, Vern Ceder
- Python Programming for the Absolute Beginner, Michael Dawson
- Python Pocket Reference, Mark Lutz
- Learning Python, Mark Lutz
- Python Essential Reference, David M. Beazley
- Python: Visual QuickStart Guide, Toby Donaldson
- Expert Python Programming: Learn best practices to designing, coding, and distributing your Python software, Tarek Ziade
- Python Programming in Context, Bradley Miller and David Ranum
- Python Power!: The Comprehensive Guide, Matt Telles
- Python Phrasebook: Essential Code and Commands, Brad Dayley
- Core Python Programming, Wesley J. Chun
- Python in a Nutshell, Alex Martelli
- Python Cookbook, Alex Martelli, Anna Ravenscroft, and David Ascher