Python for Forensicators

Dan O'Day

Who am I?

- Digital Forensic Examiner - Lake County HIDTA
- Contractor - Training Instructor - viaForensics (Linux & iPhone)
- Adjunct Instructor - Purdue University Calumet
- Got Sick of Repetitive Parsing Tasks While 4n68ing
  - Learned Python online using Google Code University and Other Resources

Course Outline

- Introduction
  - Lab 1 - Install Python, IDLE, wxPython & Text Editor
- Hands-on Introduction to Python Programming Concepts
- Parsing Text & Raw Data
- Parsing Delimited Files
- Parsing SQLite Databases
  - Lab 2 - Burner Application Data

Course Outline

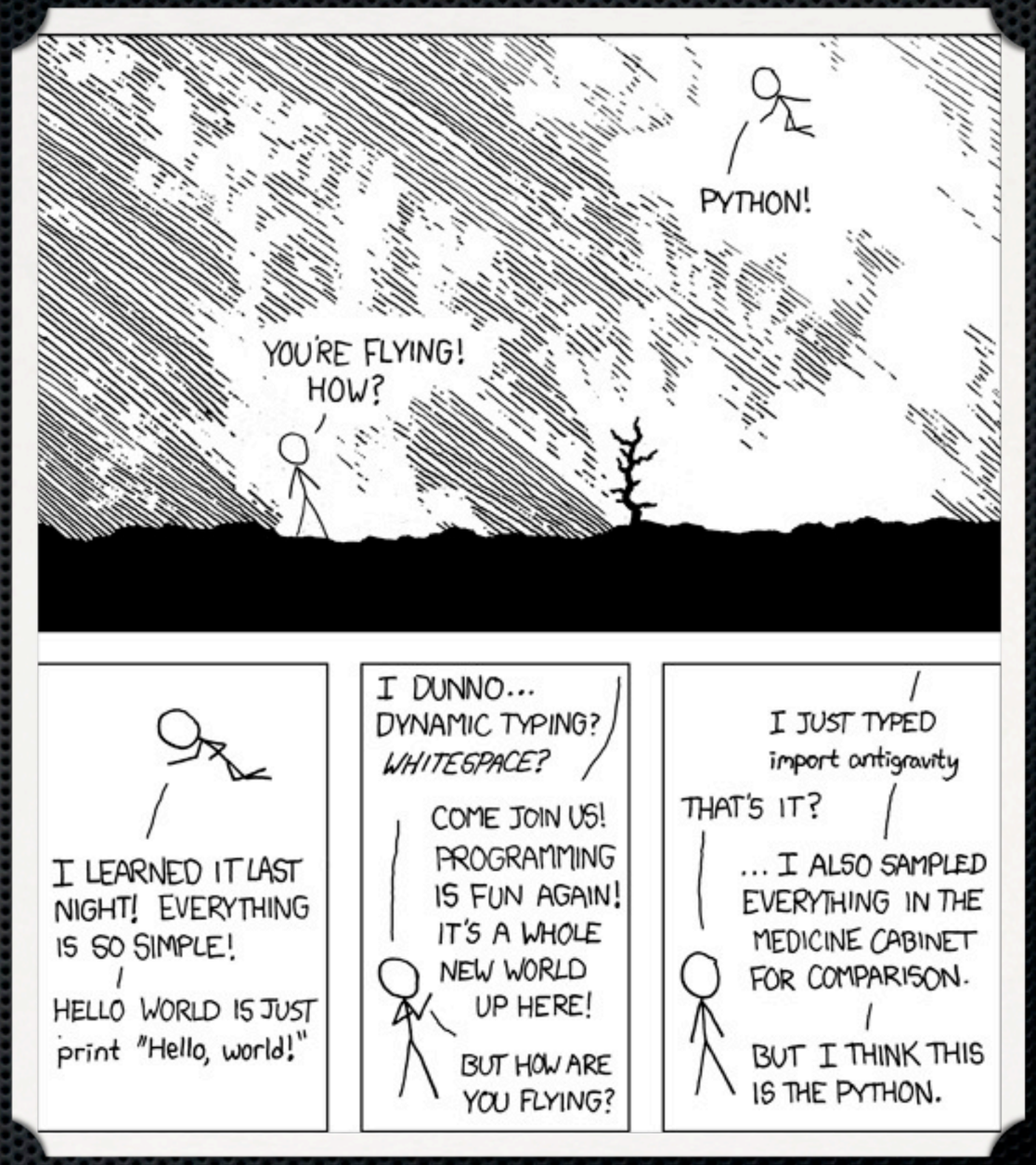
- Third-Party Libraries (time-permitting)
- Additional Resources

(Image courtesy of Jorge Cham, creator of the PhD Comics webcomic, retrieved from http://www.phdcomics.com/comics/archive/phd020507s.gif)

About Python

- Version 2 (not 3)
- Interpreted not compiled
- A lot simpler than most languages, yet very powerful
- Cross-platform compatible
- Plays well with others (.NET, Java, C/C++, etc.)
- F/OSS & great user community
- Embedded as scripting interface in popular forensics applications such as Cellebrite Physical Analyzer, XRY, X-Ways (with API plugin)

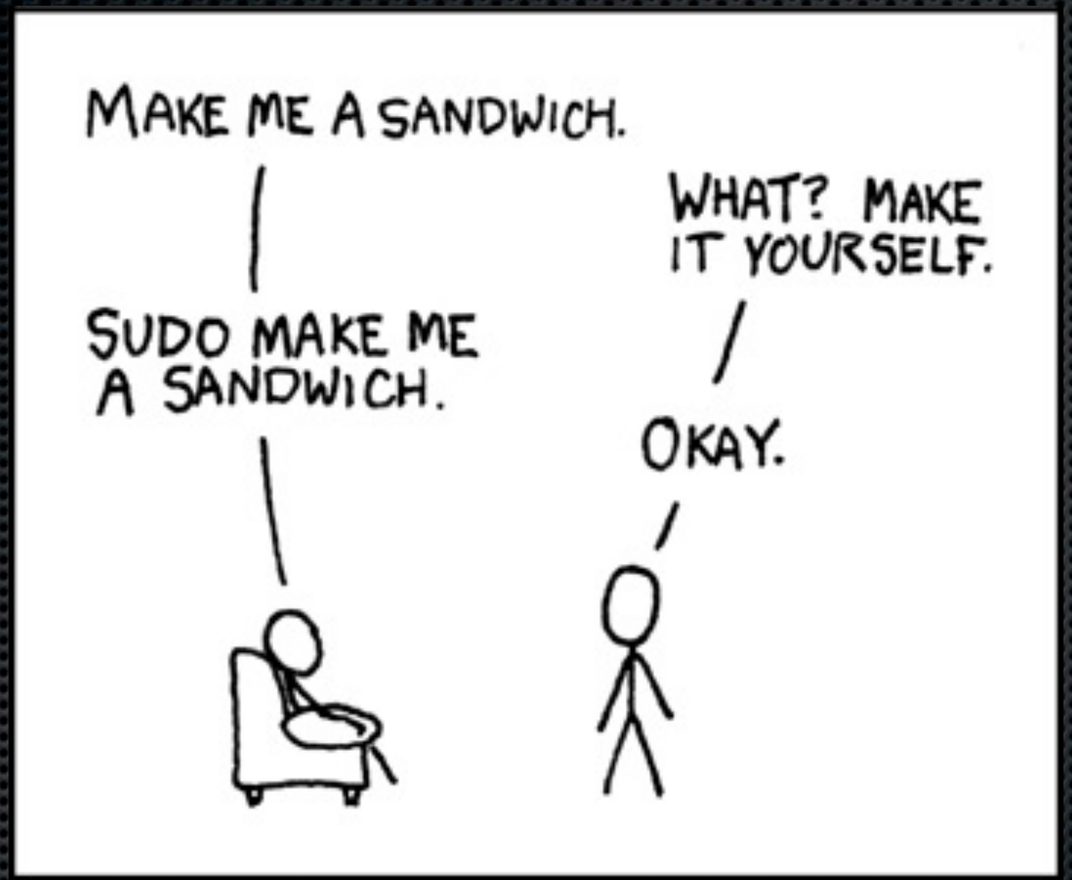
Why Learn Python?



(Image courtesy of Randall Munroe, creator of the xkcd webcomic, retrieved from https://xkcd.com/353/)

Why Learn Python?

- MUCH easier than many other languages
- Fast - fewer lines of code needed to accomplish tasks
- Automate repetitive tasks (great for parsing)
- Good documentation and user community
- Many free libraries
- Cross-platform
- Nice GUIs



Lab 1

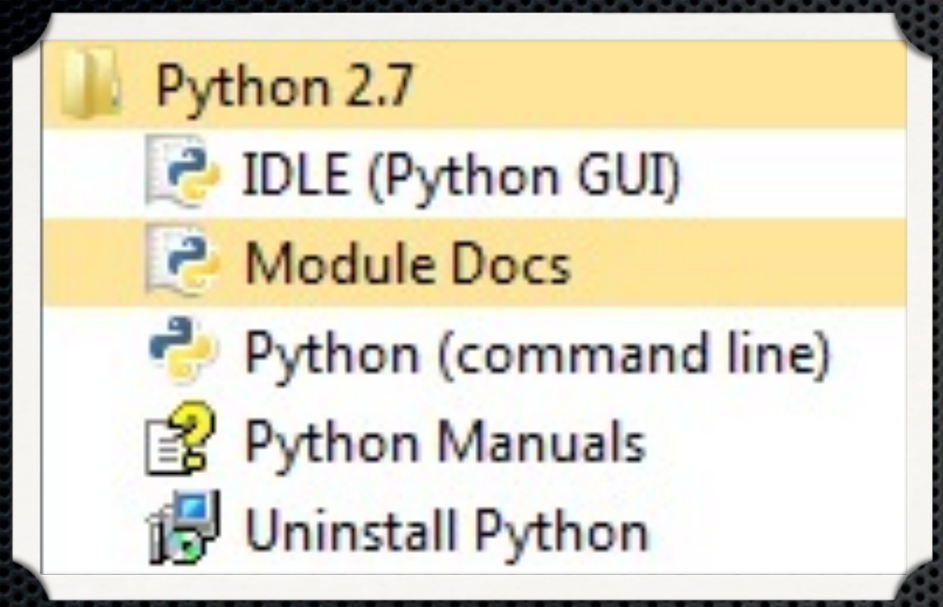
Install Python, IDLE, wxPython & Text Editor

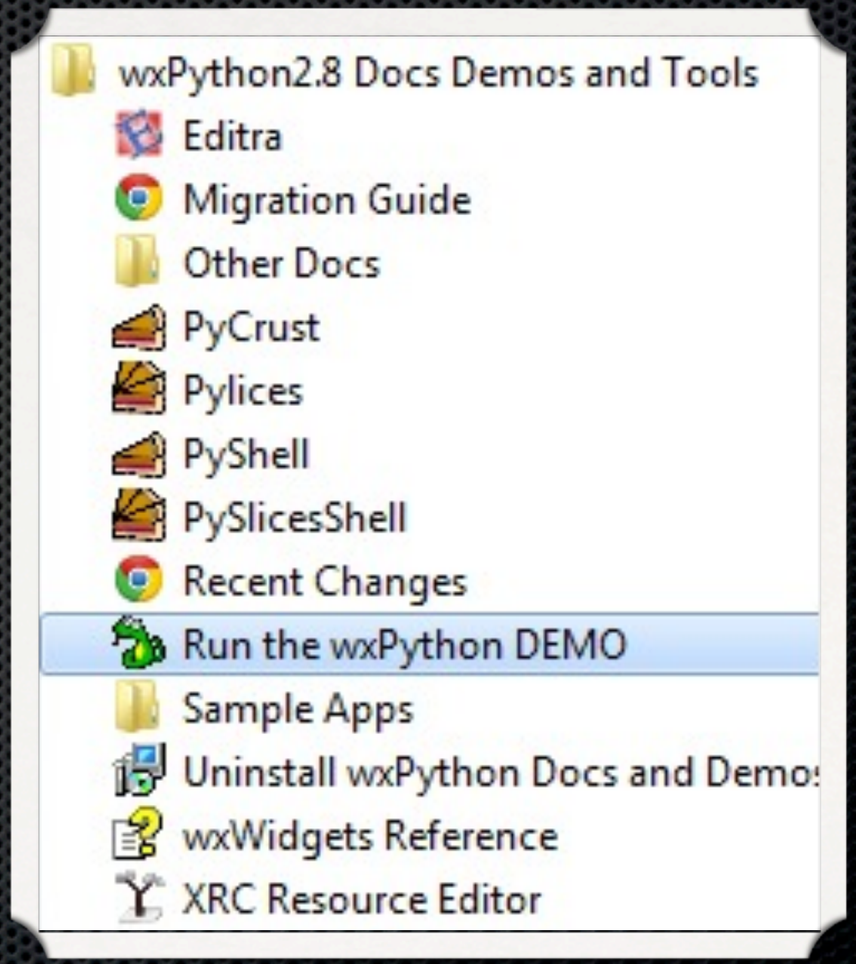
(Image courtesy of Randall Munroe, creator of the xkcd webcomic, retrieved from https://xkcd.com/149/)

:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED


Python 2.7
IDLE (Python GUI)
Module Docs
Python (command line)
Python Manuals
Uninstall Python

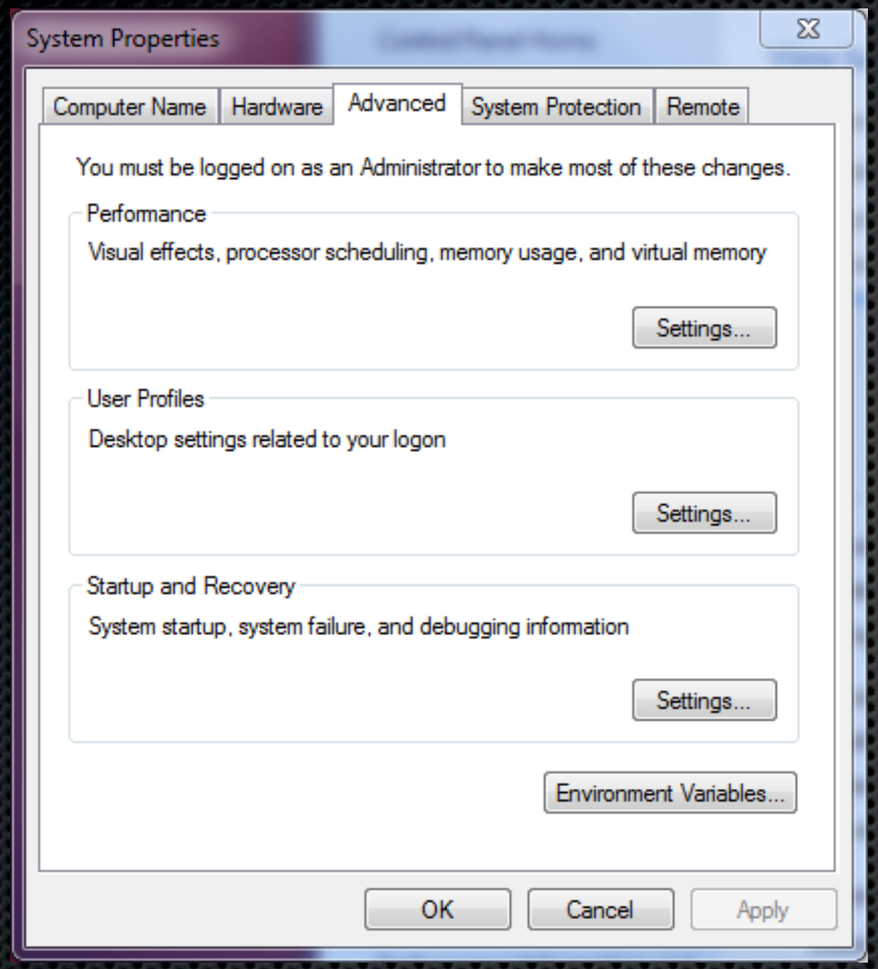

wxPython2.8 Docs Demos and Tools
Editra
Migration Guide
Other Docs
PyCrust
Pylices
PyShell
PySlicesShell
Recent Changes
Run the wxPython DEMO
Sample Apps
Uninstall wxPython Docs and Demos
wxWidgets Reference
XRC Resource Editor

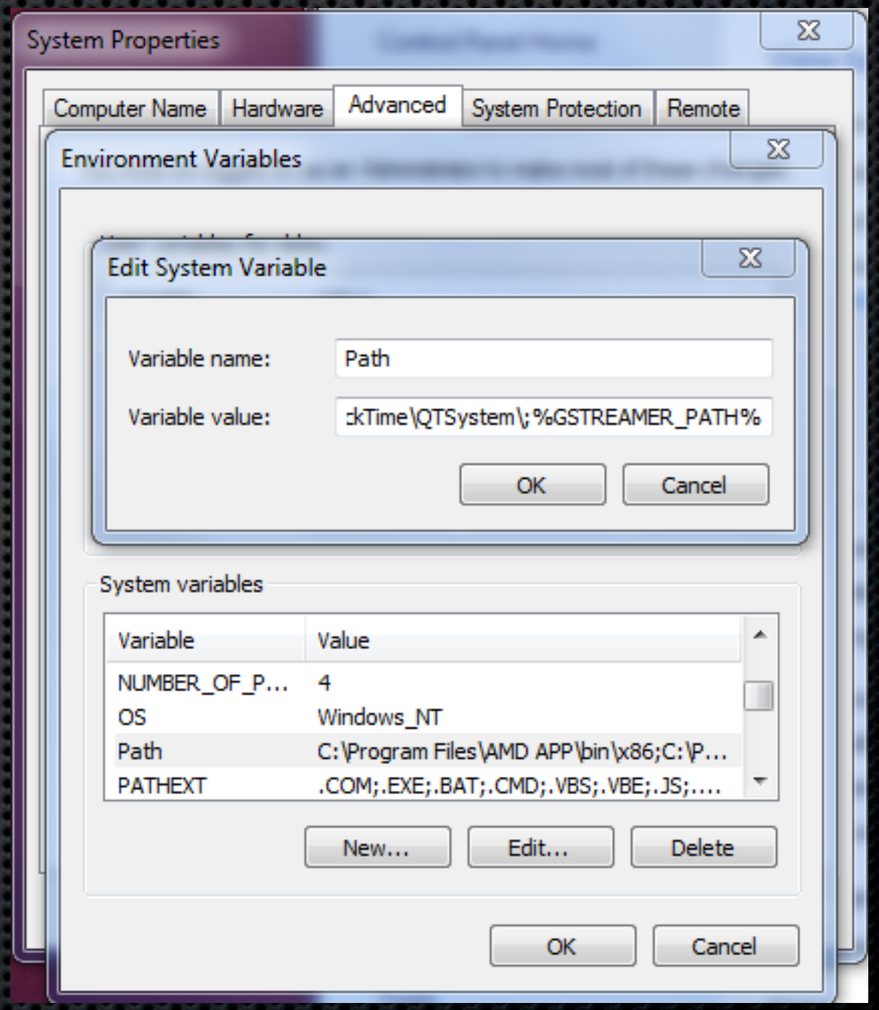
C:\Windows\system32\cmd.exe - python

```
C:\Users\dday>python -V
Python 2.7.3

C:\Users\dday>python
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

C:\Windows\system32\cmd.exe - python

```
C:\Users\dday>python
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> import wx
>>>
```


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your logon
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply

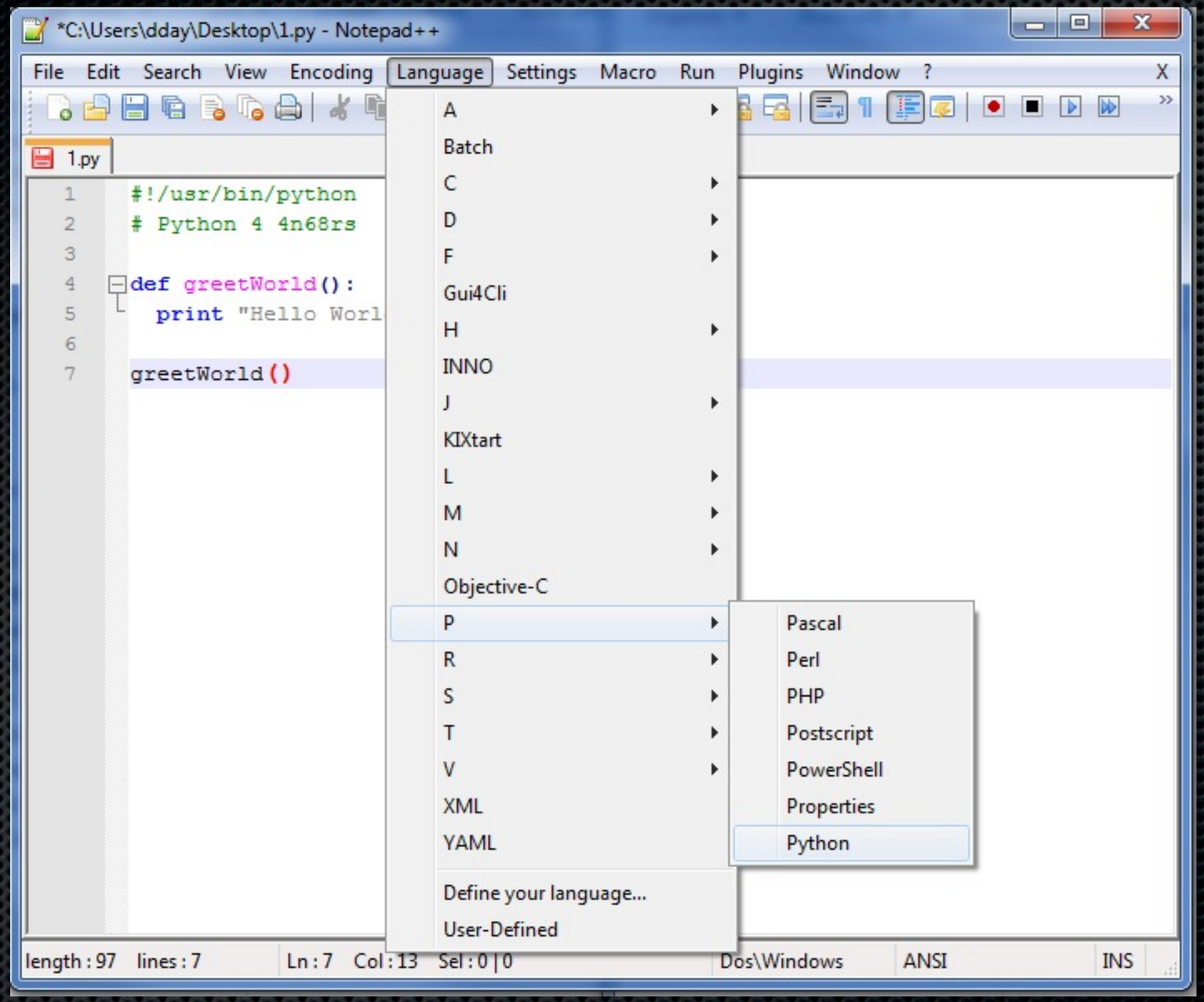

System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
Environment Variables
Edit System Variable
Variable name:
Path
Variable value:
kTime\QTSystem\;%GSTREAMER_PATH%
OK
Cancel
System variables
Variable
Value
NUMBER_OF_P...
4
OS
Windows_NT
Path
C:\Program Files\AMD APP\bin\x86;C:\P...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
New...
Edit...
Delete
OK
Cancel

Python Shell

File Edit Shell Debug Options Windows Help

```
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
```

Ln: 3 Col: 4


*C:\Users\dday\Desktop\1.py - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
1.py
#!/usr/bin/python
# Python 4 4n68rs
def greetWorld():
print "Hello Worl
greetWorld()
A
Batch
C
D
F
Gui4Cli
H
INNO
J
KIXtart
L
M
N
Objective-C
P
R
S
T
V
XML
YAML
Define your language...
User-Defined
Pascal
Perl
PHP
Postscript
PowerShell
Properties
Python
length : 97 lines : 7 Ln : 7 Col : 13 Sel : 0 | 0 Dos\Windows ANSI INS

iReplaced a PC.

danoday — Python — 80×24

```
TohuWavohu:~ danoday$ python -V
Python 2.7.3
TohuWavohu:~ danoday$ python
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

danoday — Python — 80×24

```
TohuWavohu:~ danoday$ python
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import wx
>>>
```

hw4.py

Last Saved: 5/2/12 8:11:49 PM
File Path ▾ : ~/Documents/School/PUC Technology/2012/Spring 2012/TECH581 – Biometrics for Cyber Security/Labs/Lab 10/hw4.py

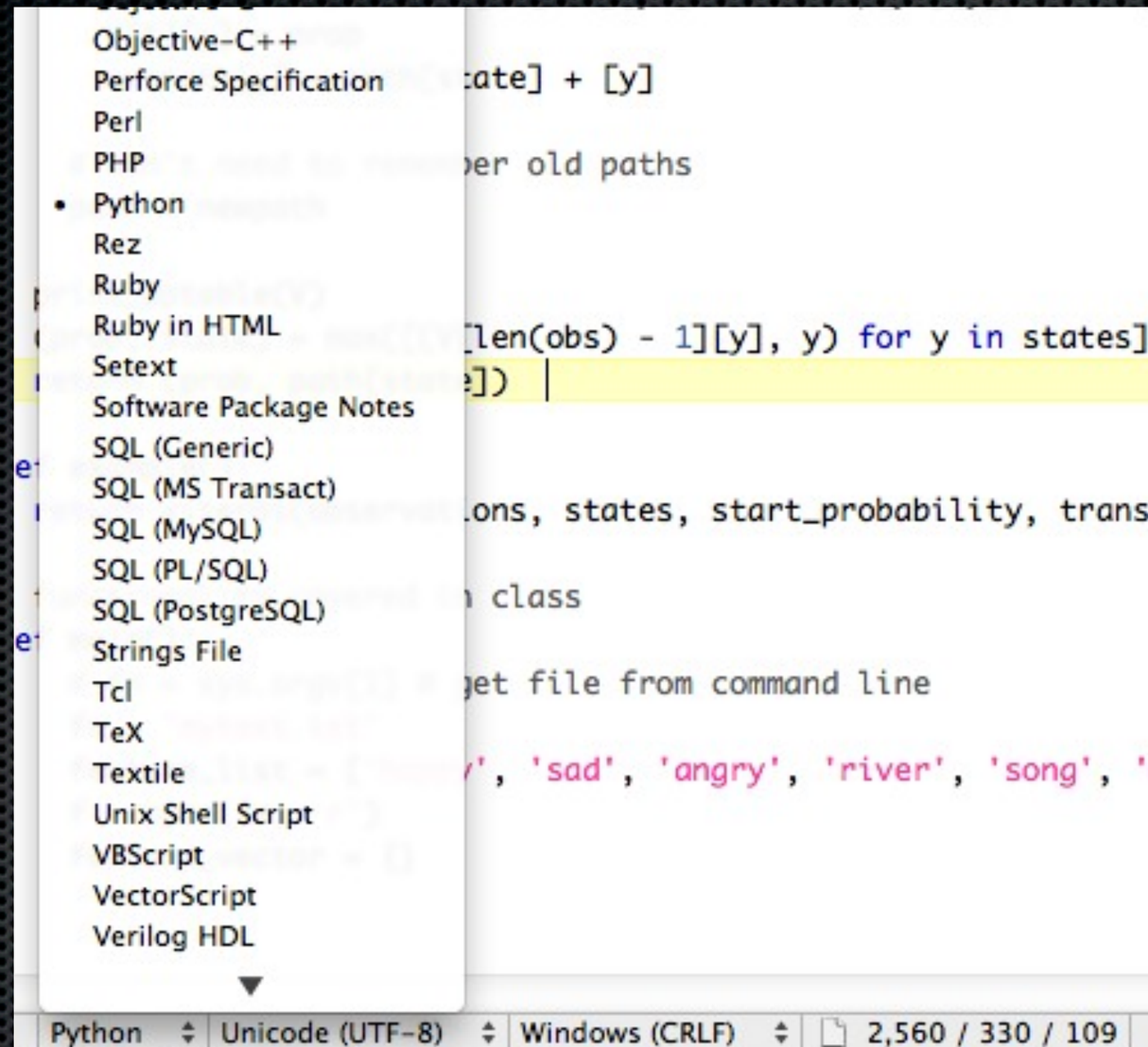
hw4.py | viterbi

```python
    print

def viterbi(obs, states, start_p, trans_p, emit_p):
  V = [{}]
  path = {}

  # initialize base cases (t == 0)
  for y in states:
    V[0][y] = start_p[y] * emit_p[y][obs[0]]
    path[y] = [y]

  # run Viterbi for t > 0
  for t in range(1,len(obs)):
    V.append({})
    newpath = {}

    for y in states:
      (prob, state) = max([(V[t-1][y0] * trans_p[y0][y] * emit_p[y][obs[t]], y0) for y0 in states])
      V[t][y] = prob
      newpath[y] = path[state] + [y]

    # don't need to remember old paths
    path = newpath

  print_dptable(V)
  (prob, state) = max([(V[len(obs) - 1][y], y) for y in states])
  return (prob, path[state])

def example():
  return viterbi(observations, states, start_probability, transition_probability, emission_probability)

# functionality covered in class
def main():
    # fn = sys.argv[1] # get file from command line
    fn = 'mytext.txt'
    feature_list = ['happy', 'sad', 'angry', 'river', 'song', 'melody', 'pond']
    f = open(fn, 'r')
    feature_vector = {}
```

Currently Open Documents
- corpus_stats.py
- followblogs.py
- gui-template.py
- hw4.py
- kiwipycon_enron_c...
- kiwipycon_import_...
- list-test.py
- nltklab.py
- parseColornoteDB.py
- pdtime.py
- prototype_parser.py
- pythonSTANford.py
- regex-test.py

Recent Documents
- hw4.py
- parseColornoteDB.py
- prototype_parser.py
- pdtime.py
- regex-test.py
- corpus_stats.py
- followblogs.py
- gui-template.py

77 | 29 | Python | Unicode (UTF-8) | Windows (CRLF) | 2,558 / 330 / 109


Objective-C++
Perforce Specification
Perl
PHP
Python
Rez
Ruby
Ruby in HTML
Setext
Software Package Notes
SQL (Generic)
SQL (MS Transact)
SQL (MySQL)
SQL (PL/SQL)
SQL (PostgreSQL)
Strings File
Tcl
TeX
Textile
Unix Shell Script
VBScript
VectorScript
Verilog HDL
ate] + [y]
er old paths
len(obs) - 1][y], y) for y in states]]
ons, states, start_probability, trans
class
et file from command line
', 'sad', 'angry', 'river', 'song',
Python
Unicode (UTF-8)
Windows (CRLF)
2,560 / 330 / 109

python44n68rs@ubuntu: ~/Desktop

```
python44n68rs@ubuntu:~/Desktop$ python -V
Python 2.7.3
python44n68rs@ubuntu:~/Desktop$ python
Python 2.7.3 (default, Sep 26 2012, 21:53:58)
[GCC 4.7.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

python44n68rs@ubuntu: ~/Desktop

```
python44n68rs@ubuntu:~/Desktop$ sudo apt-get install idle
```

jEdit - test.py

File Edit Search Markers Folding View Utilities Macros Plugins Help

File Browser

test.py (~/Desktop/)

```
def testFunction():
        print "Hello World"

testFunction()
```

2,21-28 (40/56) (python,none,UTF-8)Nmr o UG 8/15MB 7:44 PM

untu: ~/Desktop

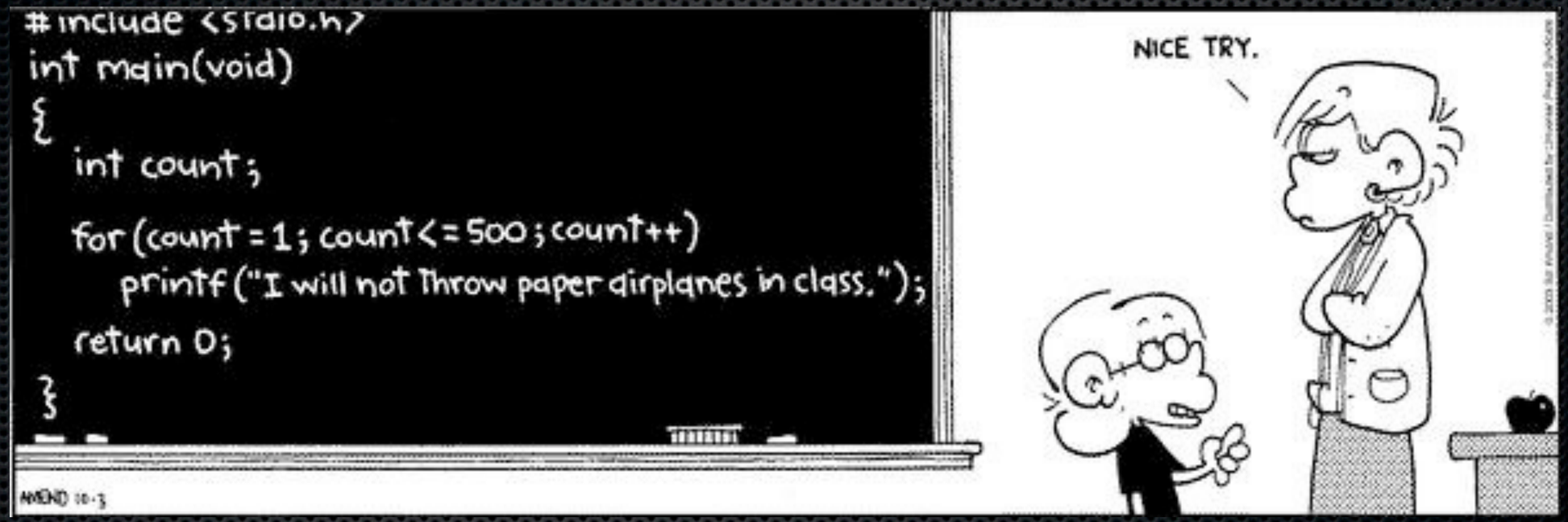
```
emacs24 &

python44n68rs@ubuntu:~$ cd ~/Desktop/
python44n68rs@ubuntu:~/Desktop$ python hello.py
HTCIA is awesome!
python44n68rs@ubuntu:~/Desktop$
```

emacs24@ubuntu

Save Undo

```
# Python 4 4n68rs

def declareAwesome(org):
    print org + " is awesome!"

declareAwesome("HTCIA")
```

Introduction to Python Programming



(Image courtesy of Bill Amend, creator of the FoxTrot Comic, retrieved from http://www.foxtrot.com/)

Python Shell

```
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print "Hello World!"
Hello World!
>>> print("Hello World! Python 3 friendly")
Hello World! Python 3 friendly
>>> |
```

Ln: 8 Col: 4

Strings

- Data type for literal text
- Enclosed in quotation marks (‘), (“), or (“”””)
- print() function can be used to display strings on screen/shell

Strings

- Syntax error = “I don’t understand you”
- Backslash (\) = literal escape

```
>>> print('My name is Dan O'Day')

SyntaxError: invalid syntax
>>> print("My name is Dan O'Day")
My name is Dan O'Day
>>>
>>> print("And he said, "Don't do that"")

SyntaxError: invalid syntax
>>> print('And he said, "Don\'t do that"')
And he said, "Don't do that"
>>>
>>> print("""Three quotes let's me do
a whole lot more
like continue onto multiple lines!""")
Three quotes let's me do
a whole lot more
like continue onto multiple lines!
>>> |
```

Variables

- A variable holds a value
- Assign values to variables using the assignment operator (=)
- (+) is used for string concatenation
- Newline (\n) and tab (\t) characters add formatting

```
>>> myVariable = "Dan O'Day"
>>> print(myVariable)
Dan O'Day
>>> print(myVariable + " is a g33k.")
Dan O'Day is a g33k.
>>>
>>> print("This is the first line.\nThis is the next.")
This is the first line.
This is the next.
>>> print("Here I am.\tNow tabbed.")
Here I am.      Now tabbed.
>>>
```

String Methods

- Methods can be called on string variables (objects)

```
>>> s = "    This is my string. There are many like it, but this one is mine.  "
>>> s.lower()
'    this is my string. there are many like it, but this one is mine.  '
>>> s.upper()
'    THIS IS MY STRING. THERE ARE MANY LIKE IT, BUT THIS ONE IS MINE.  '
>>> s.strip()
'This is my string. There are many like it, but this one is mine.'
>>> s.isalpha()
False
>>> s.isdigit()
False
>>> s.isspace()
False
>>> s.startswith('a')
False
>>> s.endswith('z')
False
>>> s.replace("is", "isn't")
"    Thisn't isn't my string. There are many like it, but thisn't one isn't mine
.  "
>>> s.split(',')
['    This is my string. There are many like it', ' but this one is mine.  ']
>>>
```

String Slices

- string[start:end]

```
String
 0  1  2  3  4  5
-6 -5 -4 -3 -2 -1
```

```
>>> s = "String"
>>> s[1:4]
'tri'
>>> s[1:]
'tring'
>>> s[:4]
'Stri'
>>> s[-1]
'g'
>>> s[-4]
'r'
>>> s[-4:]
'ring'
>>> |
```

Math Operators

- Different types of numbers (int, float)
- (+) addition (and also concatenation)
- (-) subtraction
- (*) multiplication
- (//) division
- (%) modulus (remainder)
- (**) exponent

```
>>> 5 + 30
35
>>> 22 - 6
16
>>> 5 * 18
90
>>> 6 // 2
3
>>> 7 % 3
1
>>> 1.01 + 2.5
3.51
>>> type(3.4)
<type 'float'>
>>> type(6)
<type 'int'>
```

Assignment Operators

- (=) equals (assigns value on right to variable on left)
- (+=) a += c is equivalent to a = a + c
- (-=) same as above only with subtraction
- (*=) ... multiplication
- //= ... division

Format Specifier

```
>>> print("I can't concatenate a string and an integer" + 55)

Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    print("I can't concatenate a string and an integer" + 55)
TypeError: cannot concatenate 'str' and 'int' objects
>>>
>>> s = "I have %d apples. I ate %f of them. I like %s better." % (11, 2.5, 'pears')
>>> print s
I have 11 apples. I ate 2.500000 of them. I like pears better.
>>> |
```

- (%) is also used as format specifier
- Allows variables to be inserted into strings without using concatenation operator (+)

Format Specifier

- The format specifier (%) can also convert numbers into octal or hexadecimal

```
>>> print("Geeks know that Christmas and Halloween are the same holiday because OCT 31 =
DEC 25....\nOctal: %o\nDecimal: %d\nAnd hex for good measure: %x" % (25, 25, 25))
Geeks know that Christmas and Halloween are the same holiday because OCT 31 = DEC 25....
Octal: 31
Decimal: 25
And hex for good measure: 19
>>> |
```

Unicode Strings

- Add ‘u’ prefix to make string Unicode (i18n)

```
>>> ustring = u'This is a Unicode \u018e string \xf1'
>>> print ustr
This is a Unicode string.
```

Variables Revisited

- Makes code easier to read
- Stores values for later

```
>>> pi = 3.14159265359
>>> radius = 10
>>> diameter = radius // 2
>>>
>>> area = pi * (radius ** 2)
>>> circumference = pi * diameter
>>>
>>> print area
314.159265359
>>> print circumference
15.7079632679
>>> |
```

Variable Names

- You can’t just go naming variables anything you want!
- Don’t use reserved words (and, as, break, class, continue, def, del, elif, else, except, exec, False, True, finally, for, from, global, if, import, in, is, not, none, or, pass, print, raise, return, try, while, with, yield)
- Can’t begin with numbers or special characters

Tuples ( )

- Rhymes with ‘supple’
- PARENTHESES
- Immutable
- len() function can return length of variable value

```
>>> myTuple = ("first", "second", "third")
>>> print("Joe Dimaggio rounded %s, %s, then %s base." % myTuple)
Joe Dimaggio rounded first, second, then third base.
>>>
>>> print myTuple
('first', 'second', 'third')
>>>
>>> print("But what if he only made it to %s base?" % myTuple[1])
But what if he only made it to second base?
>>>
>>>
>>> print("Joe made it around %d bases." % len(myTuple))
Joe made it around 3 bases.
```

Lists [ ]

- Changeable sequences of data
- SQUARE BRACKETS
- Methods for manipulation
- List index begins at 0 (just like strings)

```
>>> option = 0
>>> lunch = ["burger", "fries", "Coke"]
>>> print("I ate a %s for lunch." % lunch[option])
I ate a burger for lunch.
>>> option += 1
>>> print("I had some %s also." % lunch[option])
I had some fries also.
>>> option = option + 1
>>> print("I washed it all down with a %s." % lunch[option])
I washed it all down with a Coke.
>>> print("I had %d total items for lunch." % len(lunch))
I had 3 total items for lunch.
>>>
>>> lunch.append("side of ranch dressing")
>>> print lunch
['burger', 'fries', 'Coke', 'side of ranch dressing']
>>> lunch.insert(0, "candy")
>>> print lunch
['candy', 'burger', 'fries', 'Coke', 'side of ranch dressing']
>>> newList = ["watermelon", "cheesecake"]
>>> lunch.extend(newList)
>>> print lunch
['candy', 'burger', 'fries', 'Coke', 'side of ranch dressing', 'watermelon', 'cheesecake']
>>> option = lunch.index('watermelon')
>>> print("I like adding salt to my %s" % lunch[option])
I like adding salt to my watermelon
>>> print option
5
>>> lunch.remove('candy')
>>>
>>> emptyList = []
>>> emptyList.append('a')
>>> emptyList.append('b')
>>> print emptyList
['a', 'b']
>>>
>>> print lunch[1:-1]
['fries', 'Coke', 'side of ranch dressing', 'watermelon']
>>> print lunch
['burger', 'fries', 'Coke', 'side of ranch dressing', 'watermelon', 'cheesecake']
>>>
```

Dictionaries { }

- Groupings of data indexed by name
- CURLY BRACES
- Keys and values

```
>>> dinner = {}
>>> dinner["entree"] = "bambili"
>>> dinner["side"] = "corn"
>>> dinner["drink"] = "water"
>>> print dinner
{'entree': 'bambili', 'side': 'corn', 'drink': 'water'}
>>> print("I had %s for dinner with a side of %s." % (dinner["entree"], dinner["side"]))
I had bambili for dinner with a side of corn.
>>> print dinner.keys()
['entree', 'side', 'drink']
>>> print dinner.values()
['bambili', 'corn', 'water']
>>> print dinner.items()
[('entree', 'bambili'), ('side', 'corn'), ('drink', 'water')]
>>> |
```

Sorting

```
>>> listNumbers = [5, 2, 1, 3, 6, 4]
>>> print sorted(listNumbers)
[1, 2, 3, 4, 5, 6]
>>> print listNumbers
[5, 2, 1, 3, 6, 4]
>>> listNames = ['Bob', 'Jake', 'Mike', 'Alex', 'Amy']
>>> print sorted(listNames)
['Alex', 'Amy', 'Bob', 'Jake', 'Mike']
>>> print sorted(listNames, reverse=True)
['Mike', 'Jake', 'Bob', 'Amy', 'Alex']
>>>
```

Boolean Logic

- All the computer understands
  - True (1)
  - False (0)
- (==) comparison operator
  - Causes statement to be evaluated

```
>>> True
True
>>> False
False
>>> True == 1
True
>>> False == 0
True
>>> False > 0
False
>>> False < 1
True
>>> 1 == 1
True
>>> 2 + 2 == 4
True
>>> 2 + 2 == 6
False
>>> a = 'blue'
>>> b = 'black'
>>> c = 'blue'
>>> a == c
True
>>> a == b
False
>>> listA = [1, 2, 3, 4]
>>> listB = [1, 2, 3, 4]
>>> listA == listB
True
>>> |
```

Comparison Operators

- (==) checks for equality
- (!=) not equal
- (>) greater than
- (<) less than
- (>=) greater than or equal to
- (<=) less than or equal to

Logical Operators

- and
- or
- not

```
>>> a = True
>>> b = False
>>> a and b
False
>>> a or b
True
>>> not (a and b)
True
>>>
```

if Statement

- Basic decision making
- Statement evaluates T or F

```
>>> minutes = 75
>>> if minutes > 50:
        print("Time for a break!")


Time for a break!
>>> |
```

Running Python scripts inside IDLE

- File > New Window
- Type code in Window
- Save file with *.py extension
- Hit F5 to execute (Run > Run Module)

if Statement Revisited

Python Shell

```
Python 2.7.3 (v2.7.3:70274d53c1dd, Apr  9 2012, 20:52:43)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
He is buying my beverage!
>>>
```

class.py – /Users/danoday/Desktop/class.py

```
yourCoins = 1
hisCoins = 0

if yourCoins > hisCoins:
    print("He is buying my beverage!")
elif hisCoins > yourCoins:
    print("I am buying his beverage!")
else:
    print("We both will buy our own drinks.")
```

Nested if Statement

```
yourCoins = 1
hisCoins = 0

beersOnTap = ['Miller Lite', 'Coors Light', 'Blue Moon', "Bell's Oberon"]
myFavoriteBeer = "Boulevard Unfiltered Wheat"
my2ndFavoriteBeer = "Bell's Oberon"

if yourCoins > hisCoins:
    print("He is buying my beverage!")
    if myFavoriteBeer in beersOnTap:
        print("They have my favorite beer!")
    else:
        if my2ndFavoriteBeer in beersOnTap:
            print("That's OK. I also enjoy %s!" % my2ndFavoriteBeer)
        else:
            print("Oh well, I'll just drink a %s." % beersOnTap[2])
elif hisCoins > yourCoins:
    print("I am buying his beverage!")
else:
    print("We both will buy our own drinks.")
```

```
>>> ================================ RESTART
>>>
He is buying my beverage!
That's OK. I also enjoy Bell's Oberon!
>>>
```

Comments

```
# Who carries these in their pockets?
yourCoins = 1
hisCoins = 0

# Not a bad selection
beersOnTap = ['Miller Lite', 'Coors Light', 'Blue Moon', "Bell's Oberon"]

# This guy is a beer snob
myFavoriteBeer = "Boulevard Unfiltered Wheat"
my2ndFavoriteBeer = "Bell's Oberon"

# OK, who's buying the drinks?
if yourCoins > hisCoins:
    print("He is buying my beverage!")

    # Now what will I be drinking?
    if myFavoriteBeer in beersOnTap:
        print("They have my favorite beer!")
    else:
        if my2ndFavoriteBeer in beersOnTap:
            print("That's OK. I also enjoy %s!" % my2ndFavoriteBeer)
        else:
            print("Oh well, I'll just drink a %s." % beersOnTap[2])
# Shucks
elif hisCoins > yourCoins:
    print("I am buying his beverage!")
else: # looks like we both have an equal amount of coins
    print("We both will buy our own drinks.")
```

while Loop

```
condition = True
i = 10 # iterator variable

while condition == True:
    if i > 0:
        print("%d iterations remaining" % i)
    else:
        condition = False

    # decrement the iterator
    i -= 1

print("Outside of the while loop")
```

```
10 iterations remaining
9 iterations remaining
8 iterations remaining
7 iterations remaining
6 iterations remaining
5 iterations remaining
4 iterations remaining
3 iterations remaining
2 iterations remaining
1 iterations remaining
Outside of the while loop
>>>
```

for Loop

```
computers = [1, 10, 5, 4, 3, 6, 2, 2]
sumComputers = 0

for num in computers:
    sumComputers += num

print("We collectively own %d computers." % sumComputers)
```

for ... in range()

```
# print numbers 0 through 9
for i in range(10):
    print(i)

# print numbers 5 through 9
for i in range(5, 10):
    print(i)
```

Breaking Loops

- Logically, as shown previously
- OR using break statement

```
condition = True
i = 10

while condition == True:
    print(i) #print iteration
    i-=1

    # we need it to break eventually
    if i < 1:
        break
```

Importing Modules

```
import sys

print(sys.path)
```

```
>>> ================================ RESTART ================================
>>>

Traceback (most recent call last):
  File "/Users/danoday/Desktop/class.py", line 1, in <module>
    print(sys.path)
NameError: name 'sys' is not defined
>>> ================================ RESTART ================================
```

Regular Expressions

```
import re

string = """KPMG provides audit, tax & advisory services.
They work closely with their clients, helping them to mitigate risks
and grasp opportunities."""

match = re.search(r'their \w+,', string)

if match:
    print("Found '" + match.group() + "'")
else:
    print("Found nothing")
```

Regular Expressions

- a, X, 9, < -- ordinary characters just match themselves exactly. The meta-characters which do not match themselves because they have special meanings are: . ^ $ * + ? { [ ] \ | ( ) (details below)
- . (a period) -- matches any single character except newline '\n'
- \w -- (lowercase w) matches a "word" character: a letter or digit or underbar [a-zA-Z0-9_]. Note that although "word" is the mnemonic for this, it only matches a single word char, not a whole word. \W (upper case W) matches any non-word character.

(Retrieved from Google Code University: https://code.google.com/edu/languages/google-python-class/regular-expressions.html)

Regular Expressions

- \b -- boundary between word and non-word
- \s -- (lowercase s) matches a single whitespace character -- space, newline, return, tab, form [\n\r\t\f]. \S (upper case S) matches any non-whitespace character.
- \t, \n, \r -- tab, newline, return
- \d -- decimal digit [0-9] (some older regex utilities do not support but \d, but they all support \w and \s)

(Retrieved from Google Code University: https://code.google.com/edu/languages/google-python-class/regular-expressions.html)

Regular Expressions

- ^ = start, $ = end -- match the start or end of the string
- \ -- inhibit the "specialness" of a character. So, for example, use \. to match a period or \\ to match a slash. If you are unsure if a character has special meaning, such as '@', you can put a slash in front of it, \@, to make sure it is treated just as a character.
- + -- 1 or more occurrences of the pattern to its left, e.g. 'i+' = one or more i's
- * -- 0 or more occurrences of the pattern to its left
- ? -- match 0 or 1 occurrences of the pattern to its left

(Retrieved from Google Code University: https://code.google.com/edu/languages/google-python-class/regular-expressions.html)

Regular Expressions

- [ ] <-- Square brackets can be used to indicate a set of chars, so [abc] matches 'a' or 'b' or 'c'. The codes \w, \s etc. work inside square brackets too with the one exception that dot (.) just means a literal dot
- re.findall() function - returns ALL matches and returns as a list of strings

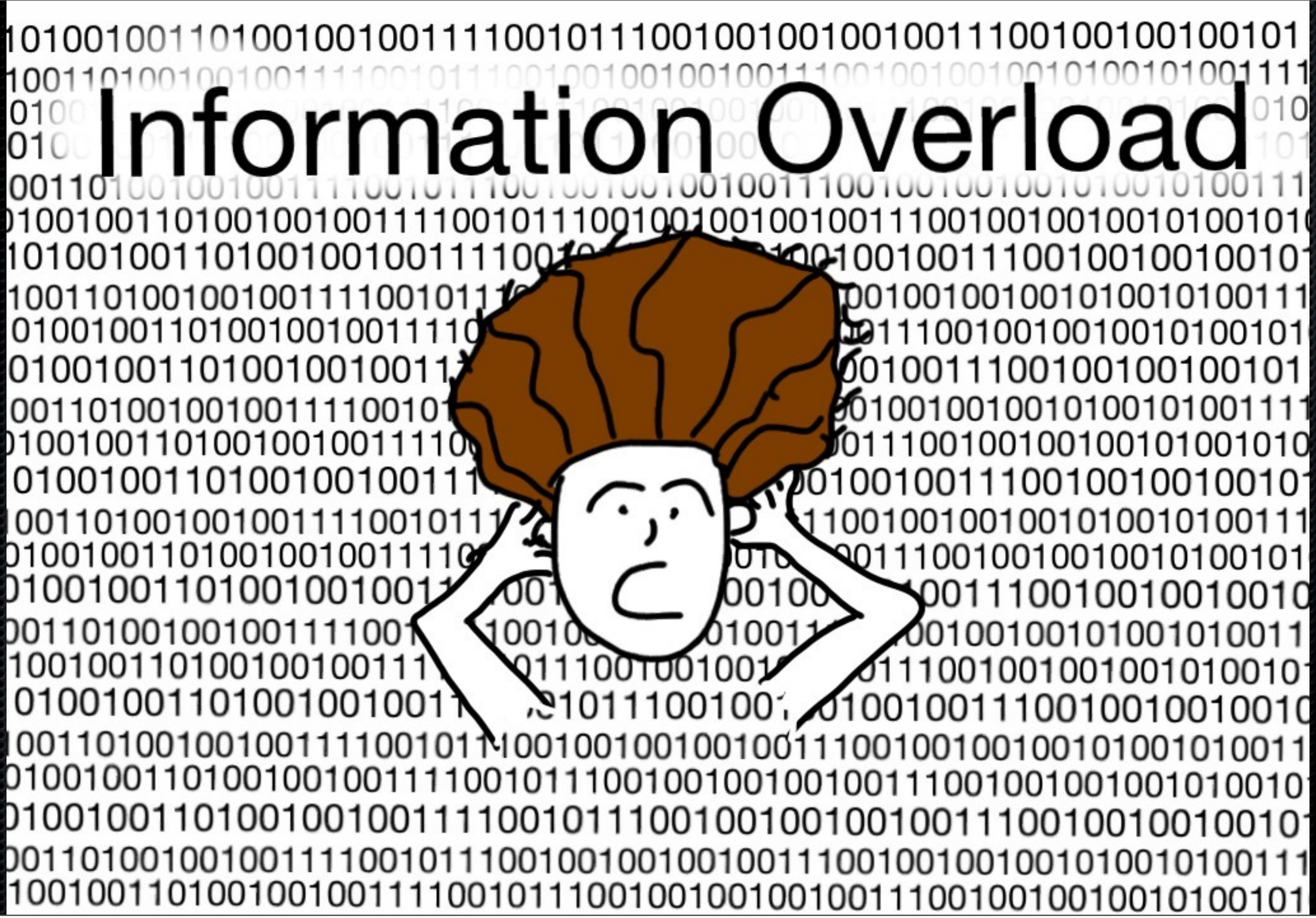
(Retrieved from Google Code University: https://code.google.com/edu/languages/google-python-class/regular-expressions.html)

Regular Expressions

```
import re

# Text with many email addresses
str = 'blah cowbell digitaloday@gmail.com tablecloth doday@me.com infosec'

# re.findall() returns a list of all the found email strings
emails = re.findall(r'[\w\.-]+@[\w\.-]+', str)
for email in emails:
    # do something with each found email string
    print email
```

```
>>>
digitaloday@gmail.com
doday@me.com
>>>
```

Information Overload



Functions

- Group code that is repeated
- Structure your program
- Reuse functions in future work
- Variable scope

```
# list tools in toolbox
def inToolbox():
    toolBox = ['duct tape', 'WD-40']
    print("I have")
    for tool in toolBox:
        if tool:
            print(tool)
            if not tool == toolBox[-1]:
                print "and"
        else:
            print("nothing")
    print("in my toolbox.")

print("What's in my toolbox?")
inToolbox()
```

Function Parameters

- Pass values to functions

```
# determine if tool is in toolbox
def inToolbox(findTool):
    toolBox = ['duct tape', 'WD-40']

    if findTool in toolBox:
        print("Yes")
    else:
        print("No")

tool2Find = 'WD-40'
print("Is there %s in my toolbox?" % tool2Find)
inToolbox(tool2Find)
```

Function Parameters

- But what if more than one person wants to use this function? What if we have different tools?

Function Parameters

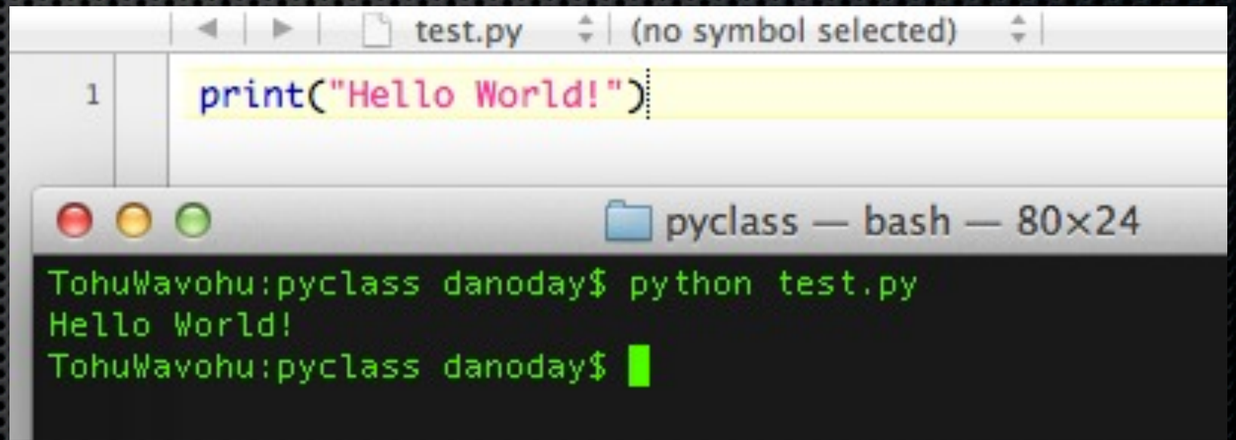
```
# determine if tool is in toolbox
def inToolbox(toolBox, findTool):
    if findTool in toolBox:
        print("Yes")
    else:
        print("No")

myToolBox = ['duct tape', 'WD-40']
tool2Find = 'WD-40'
print("Is there %s in my toolbox?" % tool2Find)
inToolbox(myToolBox, tool2Find)
```

Function Parameters

```
# return appropriate tool
def whichTool(problem):
    if problem == 'stuck':
        return 'WD-40'
    elif problem == 'loose':
        return 'duct tape'
    else:
        return 'hammer'

# determine if tool is in toolbox
def inToolbox(toolBox, issue):
    findTool = whichTool(issue)
    if findTool in toolBox:
        print("Yes")
    else:
        print("No")

myToolBox = ['duct tape', 'WD-40']
problem = 'loose'
print("Is there a tool to fix my problem?")
inToolbox(myToolBox, problem)
```

Good-bye IDLE

- Running python scripts from the command line
- TextEditor and command line interface (CLI)



Working with File Objects

- Path differences
  - Windows Users: “C:\\folder\\sample.txt”
  - Mac/Linux Users: “/folder/sample.txt”
- Import os module
- open(‘fileName.txt’, parameters)
  - r = read, w = write, a = append
  - Add U for ‘universal’ when reading (rU)

Working with File Objects

```python
import os

def makeFile(fileName):
  f = open(fileName, "w")

  # write 50 lines of text to file
  for i in range(50):
    f.write("Line %d\n" % i)
  f.close()

makeFile('test.txt')
```

Working with File Objects

```python
import os

def makeFile(fileName):
  if os.path.isfile(fileName):
    print("That file already exists!")
  else:
    f = open(fileName, "w")

    # write 50 lines of text to file
    for i in range(50):
      f.write("Line %d\n" % i)
    f.close()

makeFile('test.txt')
```

Working with File Objects

```python
import os

def appendFile(f):
  f = open(f, "a")
  f.write("\nHere is some more text!")
  f.close()
  print("Text appended")

def makeFile(fileName):
  if os.path.isfile(fileName):
    print("That file already exists!")
    appendFile(fileName)
  else:
    f = open(fileName, "w")

    # write 50 lines of text to file
    for i in range(50):
      f.write("Line %d\n" % i)
    f.close()

makeFile('test.txt')
```

Working with File Objects

```python
import os

def readFile(f):
  f = open(f, "rU")
  for line in f:
    print len(line)

def appendFile(f):
  f = open(f, "a")
  f.write("\nHere is some more text!")
  f.close()
  print("Text appended")

def makeFile(fileName):
  if os.path.isfile(fileName):
    print("That file already exists!")
    appendFile(fileName)
    readFile(fileName)
  else:
    f = open(fileName, "w")

    # write 50 lines of text to file
    for i in range(50):
      f.write("Line %d\n" % i)
    f.close()

makeFile('test.txt')
```

Working with File Objects

```
TohuWavohu:pyclass danoday$ python test.py test.txt
```

```python
import os
import sys

def readFile(f):
  f = open(f, "rU")
  for line in f:
    print len(line)

readFile(sys.argv[1])
```

Delimited Files

- Import csv module

```
import csv

def makeCSVFile():
  with open('sample.csv', 'wb') as f:
    writer = csv.writer(f)
    writer.writerow(['column1', 'column2', 'column3'])
    writer.writerow(['pie', 'jam', 'cake'])
  readCSVFile(f)
  f.close()

def readCSVFile(f):
  with open('sample.csv', 'rb') as f:
    reader = csv.reader(f)
    for col1, col2, col3 in reader:
      print col3

makeCSVFile()
```

SQLite Databases

- Structured Query Language – Light
- Database interface language
- Databases contain tables
- Tables contains records (rows of data)
- Each record has a primary key (PK) that uniquely identifies it
- Records can refer to other records using their PKs
- This is called a foreign key (FK) in this context

Basic Table - PERSON

ID (PK)	NAME	AGENCY
1	Dan O'Day	HIDTA
2	Dave Nides	KPMG
3	Mickey Mouse	Disney

New Table - AGENCY

ID (PK)	AGENCY
1	HIDTA
2	KPMG
3	HTCIA
4	Disney

Basic table revised - PERSON

ID (PK)	NAME	AGENCY_ID (FK)
1	Dan O'Day	1
2	Dave Nides	2
3	Mickey Mouse	4

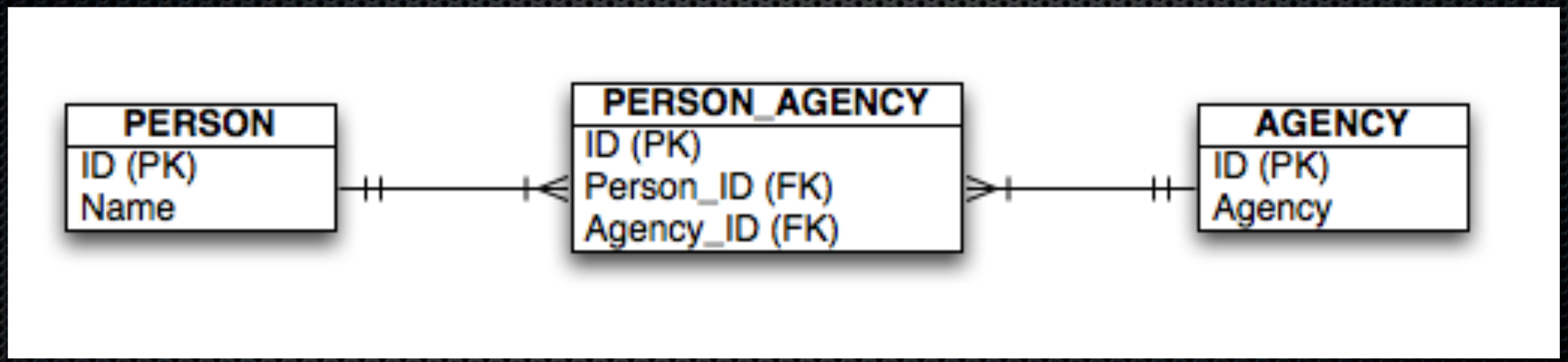
New (linking) table: PERSON_AGENCY

ID (PK)	PERSON_ID (FK)	AGENCY_ID (FK)
1	1	1
2	2	2
3	3	4
4	1	3

Basic Table Revised: PERSON

ID (PK)	NAME
1	Dan O'Day
2	Dave Nides
3	Mickey Mouse

Simplistic Entity Relationship Diagram



```
select * from PERSON
(no linking table)
```

ID (PK)	NAME	AGENCY_ID (FK)
1	Dan O'Day	1
2	Dave Nides	2
3	Mickey Mouse	4

```
select NAME from PERSON
```

NAME
Dan O'Day
Dave Nides
Mickey Mouse

```
select ID, NAME from PERSON
```

ID (PK)	NAME
1	Dan O'Day
2	Dave Nides
3	Mickey Mouse

```
select * from PERSON where NAME='Dan O' 'Day'
```

ID (PK)	NAME
1	Dan O'Day

```
select * from PERSON where ID=1
```

ID (PK)	NAME
1	Dan O'Day

```
select ID, NAME from PERSON where NAME like '%Day'
```

ID (PK)	NAME
1	Dan O'Day

```
select ID, NAME as N from PERSON
```

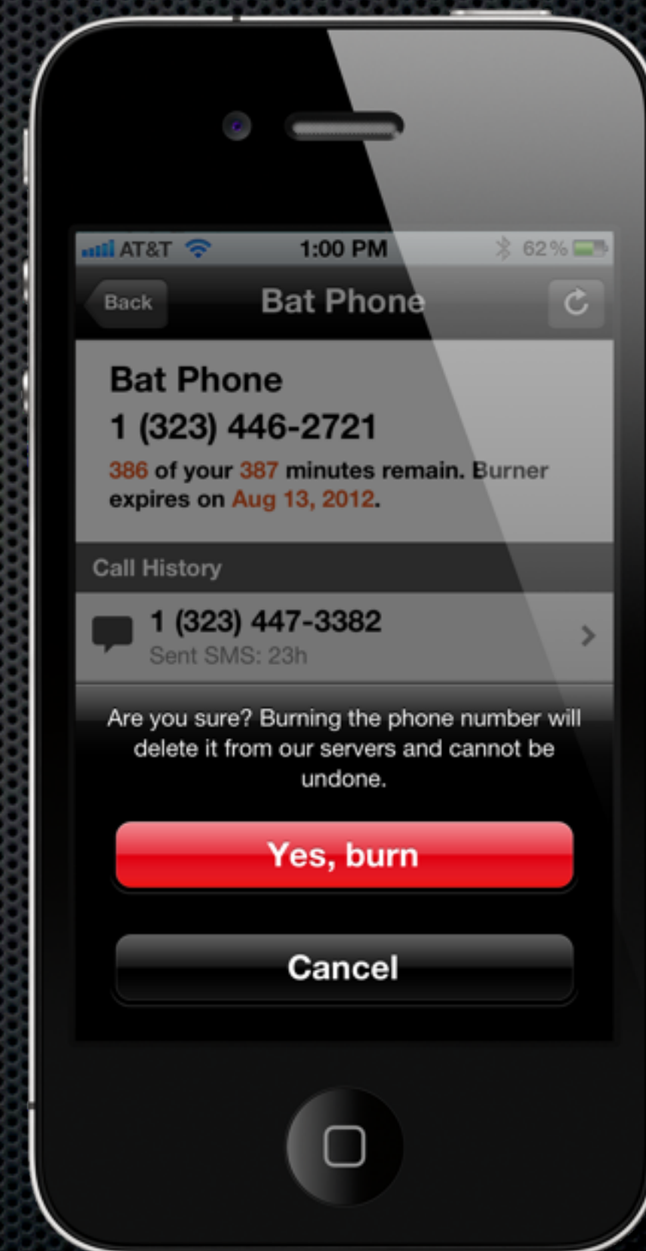
ID	N
1	Dan O'Day
2	Dave Nides
3	Mickey Mouse

```
select p.NAME, a.AGENCY from PERSON p
        inner join AGENCY a
        on p.AGENCY_ID=a.ID
```

NAME	AGENCY
Dan O'Day	HIDTA
Dave Nides	KPMG
Mickey Mouse	Disney

Parsing Timestamps in SQLite

```
select datetime(TIMESTAMP, 'unixepoch') from TABLE

select datetime(TIMESTAMP/1000, 'unixepoch') from
TABLE

select datetime(TIMESTAMP+978307200, 'unixepoch',
'localtime') from TABLE
```



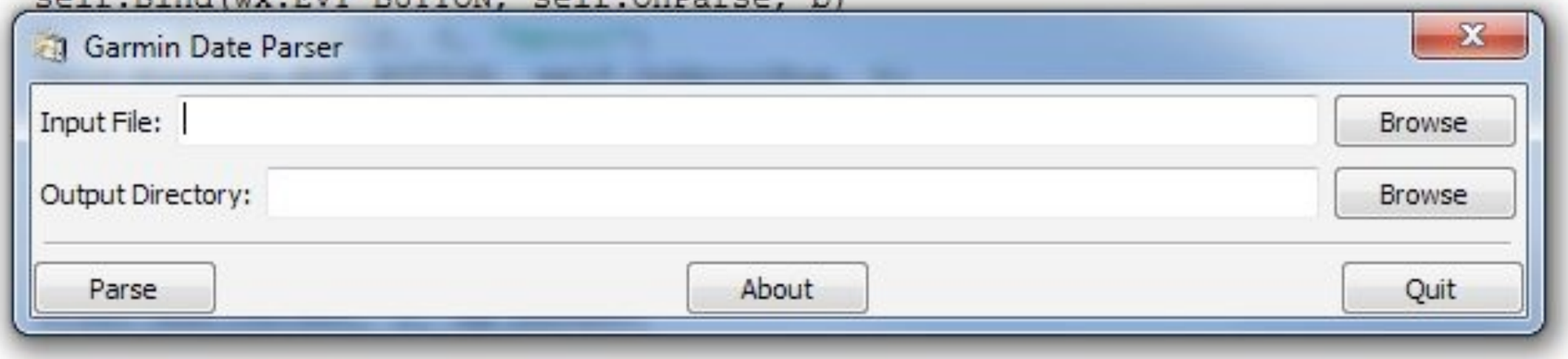
Lab 2

Parsing Burner Application Data

wxPython

- GUI Library for Python
- Cross-platform compatible
- Easy to use (relatively)

```
# [Parse] [Help] [Quit]
hsizer = wx.BoxSizer(wx.HORIZONTAL)
b = wx.Button(self, -1, "Parse")
self.Bind(wx.EVT_BUTTON, self.OnParse, b)
```




```
# Layout window
self.SetSizer(sizer)
self.SetAutoLayout(1)
sizer.Fit(self)
self.CenterOnScreen()
```

Colornote GUI App Provided

Convert Simple Scripts to GUIs

Additional Resources

- Google Code University Python Class
- Books (Violent Python, beginner books)
- Python.org
- Forums and Google (StackOverflow)