CS390-PYTHON
Programming with Python

Gustavo Rodriguez-Rivera

General Information

- Web Page: http://www.cs.purdue.edu/homes/cs390lang/python
- Office: LWSN1185
- E-mail: grr@cs.purdue.edu
- Textbook:
  - Python in a Nutshell 2nd Edition, Alex Martelli, O'REILLY

Sending your Questions

- E-mail questions to
  cs390-ta@cs.purdue.edu
- TAs office hours will be posted in the web page.

Grading

- Grade allocation
  - Final Project: 100%

Syllabus

- Installing Python
- The Python Interpreter
- The Python Language
- Exceptions
- Modules
- Core Built-ins
- Strings and Regular Expressions
- File and Text Operations
- Persistence and Databases
- CGI-Scripting

Introduction to Python

- Python is a general purpose Language
- Created by Guido van Rossum in 1990
- It is High-Level, Dynamic, Object-Oriented and Multiplatform.
- It is one of the few scripting languages that has been used successfully in large projects.
- It offers flexibility, elegance and, power.

Python Advantages

- Python programs are more compact than in other languages because:
  - – High Level Data Types allow complex operations in a single statement.
  - – Instead of using { } python uses indentation.
  - – No variables or argument declarations are necessary.

Python Standard Library

- Python also offers a Standard Library that contains useful modules for almost every need.
  - GUI
  - Databases
  - XML
  - Numerical Analysis

Installing Python

- The Python website is:

  http://www.python.org
- Go to Download->Releases->3.2 and select your platform.
- For Linux download the tar ball.
- A great tutorial about python is in:
  - http://docs.python.org/tutorial/
  - Many of the slides are based on this tutorial.

Python Interpreter

- Python takes as input a text file written in python language, compiles it and runs it.
- The executable is called “python” or “python.exe”.
- Also you can run an interactive session using the python shell with no text file.
- If you installed in Windows you can run the python interpreter:
  - Start->Python->Python (command line) or
  - Start->Python-> IDLE (Python GUI)

Environment Variables

- The execution of python is affected by the variables:
  - PYTHONHOME
    - The python installation directory
  - PYTHONPATH
    - A list of directories separated by “:” in UNIX or “;” in Windows. Modules are imported from these directories.
  - PYTHONSTARTUP
    - The name of a python source file that is executed each time an interactive session starts.

Running python

- python {options} file {arguments}
- Also in UNIX you can write a executable script that starts with the line:

  ```
  #!/usr/bin/python
  ```

  Also you have to make the script file executable.

  ```
  chmod u+x script-file
  ```

The Python Language

Lexical Structure

- Keywords

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

- Operators

  +- * / % ** // << >> & | ^ ~ < <= > >= <> != =

Literals

- 456 Integer
- 3.25 Floating Point
- 45j Imaginary Literal
- ‘String’ String Literal
- “String” String Literal that can contain ‘
- “””String””” String Literal that can contain “ and ‘

Other fundamental types

- [56, 34.5, “hi”] – List
- (34, 68, 99) – Tuple
- { ‘x’:34, ‘y’:67} - Dictionary

Comments

- Comments start with # until the end of a line

```
# this is a comment
x = 1 # this is another comment
```

Using the interpreter

- To run Python in Windows go to Start->Python->Python (command line)
- You can use the interpreter as a calculator

```
Python 3.2 (r32:88445, Feb 20 2011, 21:30:00) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 3 + 4
7
>>> 7 + 8
15
>>>
```

Variables

- Variables do not need to be defined

```
>>> x = 1
>>> y = 2
>>> x + y
3
>>>
```

String Constants

- String constants are delimited with “”

```
>>> h="Hello world"
>>> print(h)
Hello world
>>>
```

- You can also use ‘ ‘

```
>>> s = 'Hi how are you'
>>> print(s)
Hi how are you
>>>
```

String Constants

- String constants can contain new lines by using the “”” (triple double quoute) or triple single quote.

```
>>> usage = """
Usage:
command xxx yyyy
prints command
"""
>>> print(usage)

Usage:
command xxx yyyy
prints command

>>>
```

String Constants

- Two string constants are concatenated if they appear one after the other

```
>>> s = "Hello" "World"
>>> print(s)
HelloWorld
>>>
```

- To concatenate two strings even if they are not constants use the “+” operator.

```
>>> s = "Hi "
>>> t= "How are you"
>>> s + t
'Hi How are you'
>>>
```

String Indexing and Splicing

- Strings can be indexed like in C

```
>>> h="Hello world"
>>> h[0]
'H'
>>> h[5]
' '
>>> h[6]
'w‘
```

- Also strings can be sliced.

```
>>> h[2:5]
'llo'
>>>
```

- The notation a[i:j] indicates the string characters i to j-1
- If i is missing, it is assume 0. If j is missing, it is assumed the end of the string.

Strings are Immutable

- Strings cannot be modified like in C.

```
>>> h="Hello world"
>>> h[2]='g'
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    h[2]='g'
TypeError: 'str' object does not support item
   assignment
>>>
```

String Assignment

- Instead we can create a new string.

```
>>> h=h[:2] + "g" + h[3:]
>>> print(h)
Heglo world
>>>
```

Lists

- Python implements several compound types. The most useful is the “List”
- Lists are denoted with []. Example

```
>>> a=["Hi", "how", "are", "you",9]
>>> a
['Hi', 'how', 'are', 'you', 9]
```

- Like strings, lists can be sliced, concatenated etc.

```
>>> a[2]
'are'
>>>
>>> a[1:3]
['how', 'are']
>>>
```

Lists

- Unlike strings, individual elements can be modified.

```
>>> a=["Hi", "how", "are", "you",9]
>>> a
['Hi', 'how', 'are', 'you', 9]
>>> a[2]="ARE"
>>> a
['Hi', 'how', 'ARE', 'you', 9]
>>>
```

- Also elements can be removed

```
>>> a
['Hi', 'how', 'ARE', 'you', 9]
>>> a[1:3]=[]
>>> a
['Hi', 'you', 9]
>>>
```

Lists

- Or can be added anywhere in the list.

```
>>> a=["Hi", "how", "are", "you",9]
>>> a
['Hi', 'how', 'are', 'you', 9]
>>> a[1:1]=["a", "b", "c"]
>>> a
['Hi', 'a', 'b', 'c', 'how', 'are', 'you', 9]
```

- To get the length of a list use len().

```
>>> len(a)
8
```

Indexing with negative indexes

- If a negative index is used to index lists and strings, the index will be relative to the end of the list.

```
>>> a
['Hi', 'how', 'ARE', 'you', 9]
>>> a[0]
'Hi'
>>> a[4]
9
>>> a[-1]
9
>>> a[-2]
'you'
>>>
```

while statements

- The following program executes factorial:

```
>>> n=6
>>> result=1
>>> i=1
>>> while i <= n:
    result = result * i
    i = i + 1

>>> i
7
>>> result
720
```

- Notice that the block statement is indented.
- Also notice the syntax for while statement.
- As in C, the boolean expression can be a number where 0 is false and different than 0 is true.

if statements

```
if x < 0:
  print(“x is negative”)
elif x >0:
  print(“x is positive”)
else:
  print(“x is 0”);
```

- Also use indentation for multiple statements in the same block.

for statement

- The for statement is different than in C or JAVA since it is used to iterate over the elements of a list.

```
a=['Hi', 'a', 'b', 'c', 'how', 'are', 'you']
>>> for t in a:
    print (t, len(t))
Hi 2
a 1
b 1
c 1
how 3
are 3
you 3
>>>
```

The range() function

- The range function generates a list of integers inside a range.
- This function can be used to iterate over numbers.

```
>>> for t in range(5):
    print(t)

0
1
2
3
4
>>>
```

Using break, continue and else

- The ***break*** statement exits from the inner for or while loop.
- The ***continue*** statement goes to the next iteration of the loop.
- The else branch in a for statement is exectued if no **break** was executed.

```
for n in range(2, 10): # Taken from the python tutorial
   for x in range(2, n):
       if n % x == 0:
          print (n, 'equals', x, '*', n/x)
          break
       else:
          # loop fell through without finding a factor
          print (n, 'is a prime number')
```

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

Pass statement

- The pass statement does nothing. It can be used for busy waiting.

  ```
  while True:
      pass
  ```

- Also it can be used to define an empty block statement. Similar to {} in C or Java

  ```
  def foo(): # To be implemented
        pass
  ```

Defining functions

- To define function use the def() statement.

```
>>> def fact(n):
   r=1
   for t in range(1,n+1):
     r = r * t
   return r
```

Default argument values

- The arguments of a function may have default values

```
def drawLine(x1=0, y1=0, x2=50, y2=50,
   color=“green”) :


drawLine(4, 5, 6, 7) # color is green
```

- Also functions can be called using the keyword arguments

```
drawLine(x2=45, y2=90, color=“red”)
                    #It uses default values of x1, y1
```

Warning on Default Values

- The default values may keep values from previous invocations.
- This is important for mutable values like lists but not for immutable values like ints and strings.

```
>>> def f(val, list=[ ]):
        list.append(val)
    return list

>>> print(f(1))
[1]
>>> print(f(2))
[1, 2]
>>> print(f(3))
[1, 2, 3]
>>>
```

- If you do not wan this behavior default to None (no argument is passed) and initialize inside the function.

```
>>> def f(val, list=None):
        if list is None:
       list = []
        list.append(val)
    return list
```

Variable Number of Arguments

- When the last arguments in a function are *args and **keys, the arguments without keyword will be passed in *args and the arguments with keywords will be passed in **keys.

```
>>>
def elements(a, b, *args, **keys):
  print("Normal args a=",a," b=",b)
  print( "Arguments without keys:")
  for arg in args:
  print(arg)
  print("Keywords:")
  for key in keys:
  print("key=",key," val=",keys[key])
```

Variable Number of Arguments

```
>>> elements(1, 2, 3,"Hi", "everybody", color="green", day="Friday")
Normal args a= 1  b= 2
Arguments without keys:
3
Hi
everybody
Keywords:
key= color  val= green
key= day  val= Friday
>>>
```

Arguments as Lists

- Also when we have arguments in the form of a list, it is possible to make a call by passing the list as *list_of_args

```
>>> def foo(a, b, c):
    print("a=",a," b=",b," c=",c)


>>> list=[1,2,3]
>>> foo(*list)
a= 1  b= 2  c= 3
>>>
```

Coding Style

- Use 4 space indentation and no tabs.
- Wrap up line so they do not exceed 79 chars.
- Use blank lines to separate functions.
- Use CamelCase for classes and lower_case_with_undersore for functions and methods.

List Functions

- list.append(x)
  - Add item at the end of the list.
  - Similar to a[len(a):]=[x]
- list.extend(L)
  - Add list at the end
  - Siilar to a[len(a):]=L
- list.insert(i,x)
  - Insert item at a given position.
  - Similar to a[i:i]=[x]

List Functions

- list.remove(x)
  - Removes first item from the list with value x
- list.pop(i)
  - Remove item at position I and return it. If no index I is given then remove the first item in the list.
- list.index(x)
  - Return the index in the list of the first item with value x.
- list.count(x)
  - Return the number of time x appears in the list

List Functions

- list.sort()
  - Sorts items in the list in ascending order
- list.reverse()
  - Reverses items in the list.

Using Lists as Stacks

- You can use a list as a stack

```
>>> a = ["a", "b", "c“,”d”]
>>> a
['a', 'b', 'c', 'd']
>>> a.append("e")
>>> a
['a', 'b', 'c', 'd', 'e']
>>> a.pop()
'e'
>>> a.pop()
'd'
>>> a = ["a", "b", "c"]
>>>
```

Lists as Queues

- You can use a list as a queue but it is inefficient.
- For that you use queues

```
>>> from collections import deque
>>> queue = deque(["a","b","c"])
>>> queue
deque(['a', 'b', 'c'])
>>> queue.append("d")
>>> queue
deque(['a', 'b', 'c', 'd'])
>>> queue.append("e")
>>> queue.popleft()
'a'
>>> queue.popleft()
'b'
>>> queue
deque(['c', 'd', 'e'])
>>>
```

Multidimensional Arrays

- You can define a multidimensional array using lists

```
>>>
mat = [
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9],
        [10, 11, 12]
      ]
```

Multidimensional Arrays

- To iterate over the matrix

```
>>> for j in range(4):
      for i in range(3):
      print(mat[j][i])

1
2
3
4
5
6
7
8
9
10
11
12
>>>
```

The del statement

- Del can be used to remove an element of a list

```
a=[1,2,3,4]
print(a)
del a[0]
print(a)
[2,3,4]
del a[1:3]
Print(a)
[2]
```

tuples

- A tuple is another data structure in python
- A tuple consists of a number of values separated by comas
- A tuple is immutable.

```
>>> t=(1,2,3,4,5)
>>> print(t)
(1, 2, 3, 4, 5)
>>>
```

Sets

- A set is another python data structure that is an unordered collection with no duplicates.

```
>>> setA=set(["a","b","c","d"])
>>> setB=set(["c","d","e","f"])
>>> "a" in setA
True
>>> "a" in setB
False
```

Sets

```
>>> setA - setB
{'a', 'b'}
>>> setA | setB
{'a', 'c', 'b', 'e', 'd', 'f'}
>>> setA & setB
{'c', 'd'}
>>> setA ^ setB
{'a', 'b', 'e', 'f'}
>>>
```

Dictionaries

- A dictionary is a data structure that associates a key with a value.

```
>>> address={"John":"Oakwood 345", "Peter":"Evergreen 546", "Mary": "Kingston 564"}
>>> print(address["Mary"])
Kingston 564
>>> print(address)
{'John': 'Oakwood 345', 'Mary': 'Kingston 564', 'Peter': 'Evergreen 546'}
```

Dictionaries

```
>>> del address["Peter"]
>>> print(address)
{'John': 'Oakwood 345', 'Mary': 'Kingston 564'}
>>> address["Wayne"]="Young 678"
>>> print(address)
{'Wayne': 'Young 678', 'John': 'Oakwood 345', 'Mary': 'Kingston 564'}
>>> print(address.keys())
dict_keys(['Wayne', 'John', 'Mary'])
>>> "John" in address
True
```

Iterating over a dictionary

```
>>> for k in address.keys():
     print(k,":", address[k])

Wayne : Young 678
John : Oakwood 345
Mary : Kingston 564
>>>

>>> for k in sorted(address.keys()):
     print(k,":", address[k])

John : Oakwood 345
Mary : Kingston 564
Wayne : Young 678
>>>
```

Python script files

- You can write a python program outside the interpreter. Write a file ending with .py and then execute it with the python command.

fact.py:

```
def factorial(n):
    r=1
    for i in range(1,n+1):
        r = r * i
    return r

print(factorial(5))
```

```
$ Python fact.py
120
```

Passing Command Line Argument

- sys.argv contains the list of arguments passed to the script.

argecho.py:

```
import sys
for arg in sys.argv:
    print arg
```

```
%python argecho.py abc def
argecho.py abc def
```

Modules

- A module is a file written in python that has definitions that other python programs can use.
- A module can contain also statements that are executed the first time the module is loaded.

test1.py:

```
from fact import *
print(factorial(10))
```

Using Modules as scripts

- If a module is run as a script, the predefined __name__ variable is equal to “__main__”.
- We can rewrite fact.py as:

```
def factorial(n):
    r=1
    for i in range(1,n+1):
        r = r * i
    return r

if __name__ == "__main__":
  import sys
  print(factorial(int(sys.argv[1])))
```

Using Modules as scripts

```
$ python fact.py 5
120

$ python test1.py
3628800
```

PYTHONPATH

- The PYTHONPATH environment variable is a list of the directories separated by “:” (or “;” in Windows) that contains the directories with modules to be imported
- In the instruction

  from fact import *

  The fact module is searched in the directories in PYTHONPATH

Compiled Python files

- Python compiles files into bytecodes that are faster to execute.
- The compiled version of a file xyz.py is compiled into xyz.pyc and stored in the same directory where xyz is stored.
- Future runs will use the compiled version if the program has not been modified.

The dir() function

- The dir function is used to determine the functions exported by a module.

```
>>> import sys
>>> dir(sys)
['__displayhook__', '__doc__', '__excepthook__', '__name__',
    '__package__', '__stderr__', '__stdin__', '__stdout__',
    '_clear_type_cache', '_current_frames', '_getframe', '_xoptions',
    'api_version', 'argv', 'builtin_module_names', 'byteorder',
    'call_tracing', 'callstats', 'copyright', 'displayhook', 'dllhandle',
    'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix',
    'executable', 'exit', 'flags', 'float_info', 'float_repr_style',
    'getcheckinterval', 'getdefaultencoding', 'getfilesystemencoding',
    'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof',
    'getswitchinterval', 'gettrace', 'getwindowsversion', 'hash_info',
    'hexversion', 'int_info', 'intern', 'maxsize', 'maxunicode', 'meta_path',
    'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform',
    'prefix', 'setcheckinterval', 'setprofile', 'setrecursionlimit',
    'setswitchinterval', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion',
    'version', 'version_info', 'warnoptions', 'winver']
```

Packages

- Packages is a way to organize modules in python.
- The module X.Y for example designates a module Y inside package X.
- The packages can also include other packages and they will be stored in directories and subdirectories that represent this hierarchy.

Packages

- For example:

```
sound/                          Top-level package
      __init__.py                 Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
```

Packages

- The __init__.py file contains initialization code that runs the first time the package is imported.
- The statement:

  ```
  import sound.effects.echo
  ```

  Imports the echo function.
- To run one of the functions:

  ```
  sound.effects.echo.echofilter()
  ```

Packages

- It is easier to use:

```
from sound.effects.echo import echofilter
```

And then call

```
echofilter()
```

Input/Output

- The easiest way to do formatting is to format a string before printing.
- The function str(x) translates x to a string that a human can understand.
- The function repr(x) translates x to a string that the interpreter can understand.

Formatting

- Here are two ways of printing a table of squares and cubes.

```
 >>> for x in range(1,5):
   print(repr(x).rjust(2),
   repr(x*x).rjust(3), repr(x*x*x).rjust(4))

  1   1    1
  2   4    8
  3   9   27
  4  16   64
>>>
```

- rjust(n) right justifies a string by filling with spaces the left of the string so the resulting string is the size n.
- There is also str.ljust() and str.center().

Formatting

- The second way uses the format function:

```
>>> for x in range(1,5):
  print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))

 1   1    1
 2   4    8
 3   9   27
 4  16   64
>>>
```

- The substrings {pos:format} indicate that the variable at that position will be formatted in the way it is indicated.

- :2d indicates right justified using two characters.

Formatting

- Other examples:

```
>>> print('{0} and
  {1}'.format("apples","oranges“))
apples and oranges

>>> print('This {fruit} tastes
  {flavor}.'.format(fruit="strawberry",flavor="sw
  eet"))
This strawberry tastes sweet.

>>> import math
>>> print("The value of PI is
  {0:.3f}".format(math.pi))
The value of PI is 3.142
```

Reading and Writing Files

- The function f=open(filename, mode) opens a file for reading or writing.
- mode can be:
  - “r” – Reading
  - “w”- Writing
  - “a”- Append
  - “r+”- Open for both reading and writing.
  - “b” – Open the file in binary mode (only in Windows)

Reading a file

- s = f.read(size)
  - It will read size characters from the file and place it in string s.
- s = f.read()
  - It reads the entire file and places it in string s.
- s = f.readline()
  - It reads a single line from the file.
- l = f.readlines()
  - It returns a list with all the lines in the file.

Writing to a file

- f.write(s)
  - It write string s to file f.
- f.seek(pos)
  - Changes the fileposition to pos
- f.close()
  - It closes the file

Example of using Files

```
>>> f=open("f.py","r")
>>> s=f.read()
>>> print(s)

def fac(n):
    r=1
    for i in range(1,n+1):
        r = r * i
    return r

import sys
n = int(sys.argv[1])
print("The factorial of ", n, " is ", fac(n))


>>> f.close()
>>>
```

The pickle module

- Converting an object to a string representation is called “pickling” or serializing.
- Converting an object back from a string representation to an object is called unpickling or deserializing.
- pickle.dump(x,f)
  - Writes object x as a string into file f
- x=pickle.load(f)
  - Load x from string representation in file f.

Exceptions

- In python the best way to handle errors is with exceptions.
- Exceptions separates the main code from the error handling making the program easier to read.

```
try:
    statements
except:
    error handling
```

Exceptions

- Example:

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print "I/O error({0}): {1}".format(errno, strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
    raise
```

The finally statement

- The finally: statement is executed under all circumstances even if an exception is thrown.
- Cleanup statements such as closing files can be added to a finally stetement.

Finally statement

```
import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except IOError as (errno, strerror):
    print "I/O error({0}): {1}".format(errno, strerror)
except ValueError:
    print "Could not convert data to an integer."
except:
    print "Unexpected error:", sys.exc_info()[0]
finally:
     f.close() # Called with or without exceptions.
```

Predefined cleanup actions

- The ***with*** statement will automatically close the file once it is no longer in use.

```
with open("myfile.txt") as f:
  for line in f:
      print line
```

After finishing the with statement will close the file.

Running Python CGI Scripts in the CS Department

- You can make they CS webserver run your own cgi scripts.
- The information about how to setup your cgi-bin directory is in: http://support.cs.purdue.edu/help/WEB_Pages,_CGI?action=purge
- The cgi-bin runs in a separate server for security reasons.
- Login to lore and run
  - /p/www/setup-cgi

  to setup your cgi-bin directory.
- Your scripts can be added to

  /p/www-cgi/cgi-bin/*your-login*
- It can be accessed from a browser using

  http://www-cgi.cs.purdue.edu/cgi-bin/*your-login*/*your-script*

Example

- Go to lore and run

  /p/www/setup-cgi
- This will create the directory

  /p/www-cgi/cgi-bin/*your-login*
- *In this directory create a file called “python-test” and add the following content:*
-
  ```
  #!/p/python-2.5.1/bin/python
  s = """Content-type: text/html

  <H1>Hi CS390 PYTHON</H1>
  """
  print(s)
  ```
- *Make python-test executable.*

  ```
  chmod 775 python-test
  ```
- *Now you can access this script from a browser*

  http://www-cgi.cs.purdue.edu/cgi-bin/*your-login*/*python-test*

NOTES

- Remember that this cgi-bin will run in your behalf so you have to be careful.
- Do not use too much disk space.
- For the project install your own apache server and have the server serve your python requests.

Pyhton CGI Example

Create the following file in PictureShare and name it m.cgi

```
#!/usr/bin/python

import cgi
import cgitb; cgitb.enable()  # for troubleshooting

print "Content-type: text/html"
print

print """
<html>

<head><title>Sample CGI Script</title></head>

<body>

  <h3> Sample CGI Script </h3>
"""
```

Pyhton CGI Example

```
form = cgi.FieldStorage()
message = form.getvalue("message", "(no message)")

print """

  <p>Previous message: %s</p>

  <p>form

  <form method="post" action=“m.cgi">
    <p>message: <input type="text" name="message"/></p>
  </form>

</body>

</html>
""" % cgi.escape(message)
```

Example of string interpolation

- The mod (%) operator in strings is used for formatting or to insert variables in a string.

```
print 'There are %d oranges in the basket' % 32
```

```
print 'There are %d %s and %d %s in the basket' % (12,
    “oranges”,23,”apples”)
```

Using the CGI module

- See http://docs.python.org/library/cgi.html
- To use the cgi module include in your file

  import cgi

- It is recommended for debugging to import the cgitb module to print stack traces in HTML when an error happens.

  import cgitb

  cgitb.enable()

Accessing the fields

- To access the fields of a form use:

```
form = cgi.FieldStorage()
if "name" not in form or "addr" not in form:
    print "<H1>Error</H1>"
    print "Please fill in the name and addr fields."
    return
print "<p>name:", form["name"].value
print "<p>addr:", form["addr"].value
```

Upload files

- A form field may represent a file.
- To check if a field is a file and to get its content use:

```
fileitem = form["userfile"]
if fileitem.file:
    # It's an uploaded file; count lines
    linecount = 0
    while 1:
        line = fileitem.file.readline()
        if not line: break
        linecount = linecount + 1
        # do something with the file
```

Security

- When executing external commands such as:

  os.system()

  os.popen()

  do not pass arbitrary strings from the client to the shell.
- Check the arguments.
- Hackers have exploited this in the past.

Testing your cgi script.

- You can add the variable=val to your URL
- Example for the m.cgi script:

  http://sslab03.cs.purdue.edu:8081/PictureShare/m.cgi?message=Howareyou

- You can also check in the http server log files.

  ```
  tail -f logfile
  ```

age.cgi example

```
#!/usr/bin/python

# Import the CGI module
import cgi

# Required header that tells the browser how to render the HTML.
print "Content-Type: text/html\n\n"
```

age.cgi example

```
# Define function to generate HTML form.
def generate_form():
    str = """
<HTML>
<HEAD>
<TITLE>Info Form</TITLE>
</HEAD>
<BODY BGCOLOR = white>
<H3>Please, enter your name and age.</H3>
<TABLE BORDER = 0>
<FORM METHOD = post ACTION = \"age.cgi\">
<TR><TH>Name:</TH><TD><INPUT type = text  name = \"name\"></TD><TR>
<TR><TH>Age:</TH><TD><INPUT type = text name =\"age\"></TD></TR>
</TABLE>
<INPUT TYPE = hidden NAME = \"action\" VALUE = \"display\">
<INPUT TYPE = submit VALUE = \"Enter\">
</FORM>
</BODY>
</HTML>
"""
    print(str)
```

age.cgi example

```
# Define function display data.
def display_data(name, age):
    str="""
<HTML>
<HEAD>
<TITLE>Info Form</TITLE>
</HEAD>
<BODY BGCOLOR = white>
%s you are %s years old.
</BODY>
</HTML>
"""
    print(str % (name,age))
```

age.cgi example

```
# Define main function.
def main():
    form = cgi.FieldStorage()
    if (form.has_key("action") and
  form.has_key("name") and form.has_key("age")):
        if (form["action"].value == "display"):
            display_data(form["name"].value,
  form["age"].value)
    else:
        generate_form()

# Call main function.
main()
```

Structured Query Language (SQL)

- Information for web applications is usually stored into a SQL database.
- Multiple databases exist: Oracle, MySQL, Microsoft SQL, SQLite etc.
- You will be using sql lite since it is already incorporated into python.
- Databases have a GUI program that lets you manipulate the database directly and can be used for database administration.
- You can think of a database as a group of named tables with rows and columns.
- Each column has a name and each row contains a set of related data.

SQL by Example (from textbook)

Authors Table

Autor_ID	Name	Fname
ALEX	Alexander	Christopher
BROO	Brooks	Frederick P.

Books Table

Title	ISBN	Publisher_ID	Price
A Guide to the SQL Standard	0-201-96426-0	0201	47.95
A Pattern Language: Towns, Buildings, Construction	0-19-501919-9	0407	65.00

SQL by Example (from textbook)

BookAuthors Table

ISBN	Author_ID	Seq_No
0-201-96426-0	DATE	1
0-201-96426-0	DARW	2
0-19-501919-9	ALEX	1

Publishers Table

Publisher_ID	Name	URL
0201	Addison-Wesley	www.aw-bc.com
0407	John Wiley & Sons	www.wiley.com

SQL by Example

- By convention SQL keywords are written in uppercase.
- SELECT * FROM Books
  - This query returns all rows in the Books table.
  - SQL statements always require FROM
- SELECT ISBN, Price, Title

  FROM Books
  - This query returns a table with only the ISBN, price and title columns from the Books table.

SQL by Example

-
  ```
  SELECT ISBN, Price, Title
  FROM Books
  WHERE Price <=29.95
  ```
  - This query returns a table with the ISBN, Price and Title from the Books table but only for the books where the price is less or equal to 29.95.

SQL by Example

- SELECT ISBN, Price, Title

  FROM Books

  WHERE Title NOT LIKE “%n_x%”

  - Returns a table with Price and Title as columns excluding the books that contain Linux or UNIX in the title.
  - The “%” character means any zero or more characters. “_” means any single character.

SQL by Example

```
SELECT  Title, Name, URL
FROM Books, Publishers
WHERE Books.Publisher_ID=Publishers.Publisher_ID
```

It returns a table with the Title, Name of pusblisher, and URL from Books and Publishers.

Title	Name	URL
A Guide to the SQL Standard	Addison-Wesley	www.aw-bc.com
A Pattern Language: Towns, Buildings, Construction	John Wiley & Sons	www.wiley.com

SQL by Example

- You can also use SQL to change data inside a database.

-
  ```
  UPDATE Books
  SET Price = Price – 5.00
  WHERE Title Like “%C++%”
  ```

  This reduces the price by $5.00 for all books that have C++ in the title.

SQL by Example (from textbook)

- You can also delete rows with SQL
- DELETE FROM Books

  WHERE Title Like “%C++%”

  - This deletes all books that have C++ in the title.

SQL by Example

- Use INSERT to insert a new row in the table.
- INSERT INTO Books

  VALUES (‘A Guide to the SQL Standard’, ‘0-201-96426-0’, ‘0201’, 47.95)

  – This inserts a new book in the Books table.

SQL by Example

- You can also create a new table using SQL
- CREATE TABLE Books

```
 (
   TITLE CHAR(60),
   ISBN CHAR(13),
   Publisher_ID CHAR(6),
   Price DECIMAL(10,2)
)
```

Speeding Up Searches with Indexes

- You can speed up the search in a table by using Indexes.
- An index can be created in a table to find data faster.
- Updating a table with indexes is slower since the index need also be updated.
- The index is often implemented using B-trees.
- The users will not see the indexes. Indexes will just speed up the search.

Speeding Up Searches with Indexes

- To create an index in a table use this syntax.

```
CREATE INDEX index_name
    ON table_name (column_name)
```

- Duplicate values are allowed.

- If you want to create a unique index that does not allow duplicates, create a Unique Index

```
CREATE UNIQUE INDEX index_name
    ON table_name (column_name)
```

SQL Tutorials

- For more information about SQL, see the SQL tutorial in
  - http://www.w3schools.com/sql/default.asp

- You can also run some SQL examples there.

Using SQL in Python

- There is a tutorial on SQlite in:

  http://docs.python.org/2/library/sqlite3.html

Simple Picture Album Script

- Assume the following directory structure of the picture albums:
  - ~/PictureShare/users/<user>/<album>/
- For example:

```
ls -R
./users:
george@cs.purdue.edu  mary@cs.purdue.edu  peter@cs.purdue.edu

./users/george@cs.purdue.edu:
album1  album2

./users/george@cs.purdue.edu/album1:
Frozen_Fruits_Vegetable.jpg  paintings-35792675.jpg
   vangogh_spblue.jpg baby.jpg
   rose_1_bg_030703.jpg vangogh_spdark.jpg

./users/george@cs.purdue.edu/album2:

./users/mary@cs.purdue.edu:
album1

./users/mary@cs.purdue.edu/album1:
1976_steve_jobs3.jpg  steve-jobs-1984-macintosh.jpg
   steve_jobs3.jpg
```

Simple Picture Album Script

```
#!/usr/bin/python

import cgi
import cgitb; cgitb.enable()  # for troubleshooting
import glob

print "Content-type: text/html"
print

print """
```

Simple Picture Album Script

```
<html>
<head><title>Picture Album Script</title></head>

<body>

"""


# Get the user and album
form = cgi.FieldStorage()
if form.has_key("user") and form.has_key("album"):
    user = form["user"].value
    album = form["album"].value
else:
    #use default
    user = "george@cs.purdue.edu"
    album = "album1"
```

Simple Picture Album Script

```
print '<center><H1>%s:%s</H1></center>' % (user,album)

#Read all pictures
dir="users/"+user+"/albums/"+album+"/*“
pics=glob.glob(dir)

# Print pictures in a table

print """
<table border="0">
"""
```

Simple Picture Album Script

```
pics_in_row=0
for pic in pics:

    # Check if we need to start a new row
    if pics_in_row == 0:
        print "<tr>"

    print "<td>"
    print '<a href="%s">' % pic
    print '<img src="%s" width="100" height="100"><p>' % pic
    print '</a>'
    print "</td>"
    pics_in_row = pics_in_row + 1

    if pics_in_row == 5:
        print "</tr>"
        pics_in_row = 0
```

Simple Picture Album Script

```
#Close row if it is not closed
if pics_in_row > 0:
    print "</tr>"


print "</table>"

print """
</body>
</html>
"""
```

Running the Album Script

- Put the script album.cgi in PictureShare/public
- From a browser connect to:

http://host:port/PictureShare/public/album.cgi

Example:

http://sslab01.cs.purdue.edu:8085/PictureShare/public/album.cgi

IMPORTANT: Store information in Data Base instead of htdocs.

Simple example: login.cgi

```
#!/usr/bin/python

# Import the CGI, string, sys modules
import cgi, string, sys, os, re

# Required header that tells the browser how to render the HTML.
print "Content-Type: text/html\n\n"
```

Simple Example login.cgi

```
#!/usr/bin/python

# Import the CGI, string, sys modules
import cgi, string, sys, os, re, random

import cgitb; cgitb.enable()  # for troubleshooting

# Required header that tells the browser how to render the HTML.
print("Content-Type: text/html\n\n")

# Define function to generate HTML form.
def login_form():
    html="""
<HTML>
<HEAD>
<TITLE>Info Form</TITLE>
</HEAD>

<BODY BGCOLOR = white>

<center><H2>PictureShare User Administration</H2></center>

<H3>Type User and Password:</H3>
```

login.cgi

```
# Define function to generate HTML form.
def login_form():
    html="""
<HTML>
<HEAD>
<TITLE>Info Form</TITLE>
</HEAD>

<BODY BGCOLOR = white>

<center><H2>PictureShare User Administration</H2></center>

<H3>Type User and Password:</H3>

<TABLE BORDER = 0>
<FORM METHOD=post ACTION="login.cgi">
<TR><TH>Username:</TH><TD><INPUT TYPE=text NAME="username"></TD><TR>
<TR><TH>Password:</TH><TD><INPUT TYPE=password NAME="password"></TD></TR>
</TABLE>

<INPUT TYPE=hidden NAME="action" VALUE="display">
<INPUT TYPE=submit VALUE="Enter">
</FORM>
</BODY>
</HTML>
"""
    print html
```

login.cgi

```
# Define function to test the password.
def check_password(user, passwd):
    # Check that “..” or “/” are not in the user field
     if re.search("\.\.", user) or re.search("/", user):
        print "failed with .."
       return "failed"

    try:
        passwd_file = open("users/"+/homes/PictureShare/user+"/password.txt", 'r')
    except:
        #No user"
        return "failed"

    stored_password = passwd_file.readline().strip()
    passwd_file.close()
    #print "stored_password=\""+stored_password+"\""
    if (stored_password==passwd):
        return "passed"
    else:
        return "failed"
```

login.cgi

```
def display_admin_options():
    html="""
        <H1> Picture Share Admin Options</H1>
        <ul>
        <li> Create new album
        <li> Delete album
        <li> Make album public
        <li> Change pawword
        </ul>
        """
    print html
```

login.cgi

```
# Define main function.
def main():
    form = cgi.FieldStorage()
    if form.has_key("username") and form.has_key("password"):
        #Test password
        username=form["username"].value
        password=form["password"].value
        #print "You typed " + username + " and \"" + password +"\"<p>"
       if check_password(username, password)=="passed":
            display_admin_options()
        else:
            login_form()
            print "<H3><font color=\"red\">Incorrect user/password</font></H3>"

    else:
        login_form()

# Call main function.
main()
```

Using Sessions

- After login in, further pages need to check if the user has been authenticated.
- In this example the session is stored in a user/session.txt file.

Using Sessions

```
#!/usr/bin/python3.1

# Import the CGI, string, sys modules
import cgi, string, sys, os, re, random

# Required header that tells the browser how to render the HTML.
print("Content-Type: text/html\n\n")
```

Using Sessions

```
# Define function to generate HTML form.
def login_form():
    html="""
<HTML>
<HEAD>
<TITLE>Info Form</TITLE>
</HEAD>

<BODY BGCOLOR = white>

<center><H2>PictureShare User Administration</H2></center>

<H3>Type User and Password:</H3>

<TABLE BORDER = 0>
<FORM METHOD=post ACTION="login2.cgi">
<TR><TH>Username:</TH><TD><INPUT TYPE=text NAME="username"></TD><TR>
<TR><TH>Password:</TH><TD><INPUT TYPE=password NAME="password"></TD></TR>
</TABLE>

<INPUT TYPE=hidden NAME="action" VALUE="login">
<INPUT TYPE=submit VALUE="Enter">
</FORM>
</BODY>
</HTML>
"""
    print(html)
```

Using Sessions

```
# Define function to test the password.
def check_password(user, passwd):

    if re.search("\.\.", user) or re.search("/", user):
        print("failed with ..")
        return "failed"

    try:
        passwd_file = open("users/"+user+"/password.txt", 'r')
    except:
        #No user"
        return "failed"

    stored_password = passwd_file.readline().strip()
    passwd_file.close()
    #print( "stored_password=\""+stored_password+"\"")
    if (stored_password==passwd):
        return "passed"
    else:
        return "failed"
```

Using Sessions

```
def display_admin_options(user, session):
    html="""
        <H1> Picture Share Admin Options</H1>
        <ul>
        <li> <a href="login2.cgi?action=new-album&user={user}&session={session}"> Create new
      album</a>
        <li> Delete album
        <li> Make album public
        <li> Change pawword
        </ul>
        """
        #Also set a session number in a hidden field so the
        #cgi can check that the user has been authenticated
    print(html.format(user=user,session=session))

def read_session_string(user):
    session_file = open("users/"+user+"/session.txt", 'r')
    session = session_file.readline().strip()
    session_file.close()
    return session
```

Using Sessions

```
def create_session(user):
    n=20
    char_set = string.ascii_uppercase + string.digits
    session = ''.join(random.sample(char_set,n))

    #store random string as session number
    session_file = open("users/"+user+"/session.txt", 'w')
    session_file.write(session)
    session_file.close()
    return session

def check_session(form):
    print("Checking session")
    if "user" in form and "session" in form:
        print("User here")
        username=form["user"].value
        session=form["session"].value
        print("user=",username," session=",session)
        session_stored=read_session_string(username)
        print(" session_stored="+session_stored)
        if session_stored==session:
            return "passed"

    return "failed"
```

Using Sessions

```
def new_album(form):
    print("New album")
    if (check_session(form) != "passed"):
        login_form()
        print("Wrong session:", sys.exc_info()[0])
        return

    html = """
        <H1>New Album</H1>
        <TABLE BORDER = 0>
        <FORM METHOD=post ACTION="login2.cgi">
        <TR><TH>Album Name:</TH><TD><INPUT TYPE=text NAME="album"></TD><TR>
        </TABLE>
        <INPUT TYPE=hidden NAME="action" VALUE="new-album-response">
       <INPUT TYPE=hidden NAME=“user" VALUE=“{user}">
      <INPUT TYPE=hidden NAME=“session" VALUE=“{session}">
        <INPUT TYPE=submit VALUE="Enter">
        </FORM>
    """
    print(html.format(user=user,session=session))
```

Using Sessions

```
# Define main function.
def main():
#    try:
        form = cgi.FieldStorage()
        if "action" in form:
            action=form["action"].value
            print("action=",action)
            if action == "login":
                if "username" in form and "password" in form:
                    #Test password
                    username=form["username"].value
                    password=form["password"].value
                    #print("You typed " + username + " and \"" + password +"\"<p>")
                    if check_password(username, password)=="passed":
                        session=create_session(username)
                        display_admin_options(username, session)
                    else:
                        login_form()
                        print("<H3><font color=\"red\">Incorrect user/password</font></H3>")
            elif action == "new-album":
                print("Here1")
                new_album(form)


        else:
            login_form()
    #except:
#        login_form()
#        print("Unexpected error:", sys.exc_info()[0])

# Call main function.
main()
```

Using Sessions

- The session id can store in the URL or in the cookies.
- An alternative to using sessions is to include the user and password in hidden fields in the form in subsequent documents and verify the user and password in every operation.
- The password can be envrypted with md5 or use shtp.

Uploading files

- Assume the following upload.html file that can be generated by a CGI

```
<HTML>

<FORM ACTION=“upload.cgi" METHOD="POST"
   enctype="multipart/form-data">
   <input type="hidden" name="user" value="lola">
   <input type="hidden" name="action" value="upload">
   <BR><I>FILE:</I> <INPUT TYPE="FILE" NAME=upfile>
<br>
<input type="submit" value="Press"> to upload the file!
</form>

</HTML>
```

Uploading files

- The corresponding python file upload.cgi

File:action.py

```
#!/usr/bin/python

import cgi
import sys

def gen_html_header() :
    print "Content-Type: text/html\n\n"
    print "<HTML>"

def gen_html_trailer() :
    print "</HTML>"
gen_html_header()
form = cgi.FieldStorage()
try :
    file_contents = form["upfile"].value
    print file_contents
except :
    print sys.exc_info()

gen_html_trailer()
```

A More Complex Example to Upload Files

```
def upload_pic(form):
    if (check_session(form) != "passed"):
        login_form()
        print("Wrong session:", sys.exc_info()[0])
        return

    html="""
        <HTML>

        <FORM ACTION="login2.cgi" METHOD="POST" enctype="multipart/form-
   data">
            <input type="hidden" name="user" value="{user}">
            <input type="hidden" name="session" value="{session}">
            <input type="hidden" name="action" value="upload-pic-data">
            <BR><I>Browse Picture:</I> <INPUT TYPE="FILE" NAME="file">
            <br>
            <input type="submit" value="Press"> to upload the picture!
            </form>
        </HTML>
    ""“
    user=form["user"].value
    session=form["session"].value
    print(html.format(user=user,session=session))
```

A More Complex Example to Upload Files

```
def upload_pic_data(form):
    #Check session is correct
    if (check_session(form) != "passed"):
        login_form()
        print("Wrong session.")
        return

    #Get file info
    fileInfo = form['file']

    #Get user
    user=form["user"].value

    # Check if the file was uploaded
    if fileInfo.filename:
        # Remove directory path to extract name only
        fileName = os.path.basename(fileInfo.filename)
        open('users/'+user+"/albums/album1/" + fileName, 'wb').write(fileInfo.file.read())
        print('<H2>The picture ' + fileName + ' was uploaded successfully</H2>')
        print('<image src="users/'+user+'/albums/album1/' + fileName + '">')
    else:
        message = 'No file was uploaded'
```

A More Complex Example to Upload Files

```
# Define main function.
def main():
    #try:
        form = cgi.FieldStorage()
        if "action" in form:
            action=form["action"].value
            print("action=",action)
            if action == "login":
                 login()
            elif action == "new-album":
                new_album(form)
            elif action == "upload-pic":
                upload_pic(form)
            elif action == "upload-pic-data":
                upload_pic_data(form)

        else:
            login_form()
    #except:
    #    login_form()
    #    print("Unexpected error:", sys.exc_info()[0])

# Call main function.
main()
```

Python Web Frameworks

- There are python web frameworks that make it easy to implement web applications.
  - Django – Developed for a News web site. Very popular scalable and adaptable
  - Pylon – Influenced by Ruby on Rails.
  - TurboGears – Lightweight framework
- Django seems to be the easiest to use and has a great tutorial.
- If you want to build a solid, scalable web site in python I strongly encourage you to use a web framework.
- Google for any of them.

Classes

- The simplest class definition has the following syntax:

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

- The statements usually are function definitions but they can be any other statement

Class Example

- Assume the following class definition:

```
class MyClass:
    """A simple example class"""
    i = 12345
    def f(self):
        return 'hello world‘
```

- Here the class defines a class attribute MyClass.i and a function Myclass.f
- Notice that MyClass.i is a class variable or “static” variable.

Class instantiation

- To instantiate an object of the previous class

```
X = MyClass()
```

- This command creates an instance of MyClass and assigns it to x.
- If you need a “constructor” for your class to initialize the object, you can define the special function __init__()

```
>>> class Complex:
...   def __init__(self, realpart,imagpart):
...         self.r = realpart
...         self.i = imagpart
...
>>> x = Complex(3.0, -4.5)
>>> x.r, x.i
(3.0, -4.5)
```

- In this case self.r and self.i are instance variables.

Adding instance vars to an object

- In the previous object x, we can add more instance variables to x by assignment.

```
x.rad = 89
x.s=“Hello”
```

- To call a method:

```
x.f()
```

- Also we could call a function by reference

```
xf = x.f
xf()
```

Notes on objects

- Data attributes override method attributes in a class, causing bugs. Design a convention to prevent collisions like prefix data attribute names with an underscore (_atr).
- Potentially the users of the objects may store their own attributes in the objects themselves, causing problems. This has to be avoided.
- The first argument in a method should be ***`self`*** to represent the object instance.

Another example of a class

```
class Bag:
    def __init__(self):
        self.data = []
    def add(self, x):
        self.data.append(x)
    def addtwice(self, x):
        self.add(x)
        self.add(x)
```

Inheritance

- The syntax for inheritance is the following:

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

- BaseClassName is a previously defined class.
- During a method call, the subclass is checked first for the method, and then goes up to the parent and so on.
- Derived classes may override methods of the parent classs all methods are virtual.

Functions to check inheritance

- ***isinstance(obj, class)***
  - Returns true if ***obj*** is instance of ***class***.
- ***issubclass(class1, class2)***
  - Returns true if ***class1*** is a subclass of ***class2***.

Private Variables

- To add a private variable to an object, prepend the string “__” (double underscore) to the variable like in __x.
- Python will add internally the string _classname to the variable to become _classname__x.
- In this way it will avoid collisions.

Empty classes

- Sometimes it is handy to use a class as a struct to store a collection of attributes.

```
class Employee:
    pass

john = Employee() # Create an empty employee record

# Fill the fields of the record
john.name = 'John Doe'
john.dept = 'computer lab'
john.salary = 1000
```

The os Module

- The OS module contains functions used to interact with the Operating System.

```
import os

d = os.getcwd() # Gets current directory

os.chdir(“../xyz”) # Changes current directory

os.system(“mkdir mydir”) #Executes a shell command

dir(os) # returns the functions in OS module

help(os) # Returns a manual about the os module.
```

The shutil module

- The shutil contains functions for file manipulation

```
import shutil

shutil.copyfile(‘data.db’, ‘archive.db’)

shutil.move('/build/executables', 'installdir')
```

File wildcards

- The glob module can be used to obtained the files matched by a wildcard.

```
import glob
glob.glob("*.py")
['__init__.py', 'manage.py',
  'settings.py', 'urls.py']
>>>
```

Using sqlite3 in Python

- Python includes a simple database implementation called sqlite3 that provides a database with fast disk access.
- These presentation is based on the tutorial provided in:

  http://docs.python.org/library/sqlite3.html
- To use the database you need to create a connection object first:

```
 import sqlite3
conn = sqlite3.connect('/tmp/example')
```

- If you want the database to be created in memory use “:memory:” instead but your data will not be saved.

Using sqlite3 in Python

- Once you have a connection you can create a Cursor and use it to execute SQL commands

```
c = conn.cursor()

# Create table
c.execute('''create table stocks
(date text, trans text, symbol text,
 qty real, price real)''')

# Insert a row of data
c.execute("""insert into stocks
          values ('2006-01-05','BUY','RHAT',100,35.14)""")

# Save (commit) the changes
conn.commit()

# We can also close the cursor if we are done with it
c.close()
```

Using sqlite3 in Python

- It is not recommended to use string functions to assemble your SQL commands since it makes your program vulnerable.
- Instead use sqlite3 parameter substitution.

```
# Never do this -- insecure!
symbol = 'IBM'
c.execute("... where symbol = '%s'" % symbol)

# Do this instead
t = (symbol,)
c.execute('select * from stocks where symbol=?', t)

# Larger example
for t in [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
          ('2006-04-05', 'BUY', 'MSOFT', 1000, 72.00),
          ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
         ]:
    c.execute('insert into stocks values (?,?,?,?,?)', t)
```

Using sqlite3 in Python

- To get the data after executing SELECT, you can use the cursor as an iterator or use fetchall() to get a list of the matching table entries.

```
>>> c = conn.cursor()
>>> c.execute('select * from stocks order by price')
>>> for row in c:
...    print row
...
(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
(u'2006-04-06', u'SELL', u'IBM', 500, 53.0)
(u'2006-04-05', u'BUY', u'MSOFT', 1000, 72.0)
>>>
```

See `http://docs.python.org/library/sqlite3.html` for more information.

Regular Expressions

- To use regular expressions import the “re” package.
- A regular expression is a special string that describes a group of one or more strings.
- The simplest expression like “test” will match only the “test” string.
- If other special characters are included in the expression such as “. ^ $ * + ? { } [ ] \ | ( )” then the regular expression may describe 0 or more other strings.

The […] character class

- The [and] are used for specifying a character class.
- For example, [abc] will match the characters a, b, or c.
- When using ^ as the first character, like in [^abc] then it will mean all characters that are –not- a, b, nor c.
- The \ is used as a escape sequence. For example, \[will match the [character.

Predefined Character Classes

- Here are some predefined sets:

\d

Matches any decimal digit; this is equivalent to the class [0-9].

\D

Matches any non-digit character; this is equivalent to the class [^0-9].

\s

Matches any whitespace character; this is equivalent to the class [\t\n\r\f\v].

\S

Matches any non-whitespace character; this is equivalent to the class [^ \t\n\r\f\v].

\w

Matches any alphanumeric character; this is equivalent to the class [a-zA-Z0-9_].

\W

Matches any non-alphanumeric character; this is equivalent to the class [^a-zA-Z0-9_].

Repeating Strings with “*” and “+”

- To represent the repetitions of a string you use the “*” operator.
- “x*” will represent the repetition of “x” 0 or more times like “x” or “xx” or “”
- For example “ab*c” will match the strings “ac”, “abc”, “abbbbc”.
- The “+” operator can be used to represent 1 or more repetitions of a string.
- For example “x+” will represent 1 or more repetitions of “x”, like “x”, “xx” but not “”.

The “?” and “{}” operators.

- The “?” operators macthes a regular expression 0 or 1 time.
- For example “x?” will match “x” or “”.
- Also, “ab?c” will match “abc or “ac”.
- The “x{m,n}” qualifier represents that x can be repeated at least m times and at most n times.
- For example x{1, 3} will match “x”, “xx”, “xxx” but it will not match “” or “xxxx”.
- You can omit m or n that means that m=0 or n is infinite.
- x{2,} matches “xx”, “xxx”, “xxxx….”

Compiling Regular Expressions

- To execute faster the matching of regular expressions, they have to be compiled.
- The compilation generates a “Finite State Automaton” (FSM) that is usually represented by a matrix that allows the execution of the matching operation in linear time.

```
>>> p = re.compile('ab*c')
>>> print(p.match("a"))
None
>>> print(p.match("ac"))
<_sre.SRE_Match object at 0x00000000020FA718>
>>> print(p.match("abc"))
<_sre.SRE_Match object at 0x00000000020FA718>
>>> print(p.match("abbc"))
<_sre.SRE_Match object at 0x00000000020FA718>
>>> print(p.match("abd"))
None
>>>
```

Raw String Notation

- Sometimes the strings you want to match contain “\” characters.
- Since “\” is a special character, you need to escape it as in “\\”.
- To avoid the cascade of “\\” characters, the raw strings or r”…” allow the “\” as a character itself.
- For example:

Regular String	**Raw string**
`"ab*“`	`r"ab*”`
`“\\\\section”`	`r”\\section”`
`“\\w+\\s+\\1”`	`r"\w+\s+\1"`

Using Regular Expressions

- Once you have compiled regular expressions you can use them for matching or replacing.

match()

Determine if the RE matches at the beginning of the string.

search()

Scan through a string, looking for any location where this RE matches.

findall()

Find all substrings where the RE matches, and returns them as a list.

finditer()

Find all substrings where the RE matches, and returns them as an iterator.

Testing matching

- You can use the result of a match as the test in a if or while statement.

```
>>> p = re.compile('ab*c')
>>> s="abbbc"
>>> if p.match(s):
...     print("Matches!")
...
Matches!
>>>
>>> s="abbbbd"
>>> if p.match(s):
...     print("Matches!")
...
>>>
```

Testing matching

- Also notice that in match() the regular expression has to match the beginning of the string to succeed.

```
>>> s="xabbbc"
>>> if p.match(s):
...     print("Matches!")
...
>>>
```

- However, in search() only a substring needs to match to succeed.

```
>>>>>> if p.search(s):
...     print("Search found!")
...
Search found!
>>>
```

The MatchObject

- Also, you can assign the result of a match to a MatchObject and use it for obtaining the matched string.

```
>>> import re
>>> p = re.compile('[a-z]+')
>>> p
<_sre.SRE_Pattern object at 0x00000000021BC718>
>>> print(p.match(""))
None
>>> m = p.match('tempo')
>>> print(m)
<_sre.SRE_Match object at 0x00000000021BA718>
>>> m.group()
'tempo'
>>> m.start(),m.end()
(0, 5)
>>>
```

MatchObject Operations

- group()
  - Return the string matched by the RE
- start()
  - Return the starting position of the match
- end()
  - Return the ending position of the match
- span()
  - Return a tuple containing the (start, end) positions of the match

Testing for the MatchObject

```
p = re.compile( ... )
m = p.match( 'string goes here' )
if m:
    print 'Match found: ', m.group()
else:
    print 'No match'
```

Using findall

```
>>> p = re.compile('\d+')
>>> p.findall('12 drummers drumming, 11 pipers piping, 10 lords a-leaping')
['12', '11', '10']
```

Iterating over the finds

```
>>> iterator = p.finditer('12 drummers drumming,
   11 ... 10 ...')
>>> iterator
<callable-iterator object at 0x401833ac>
>>> for match in iterator:
...     print match.span()
...
(0, 2)
(22, 24)
(29, 31)
```

Server Sockets

- Python has a powerful Socket library that is very easy to use.
- To use it import SocketServer
- Then define a class that inherits from `SocketServer.BaseRequestHandler` and overide the method handle.
- Then call the method `SocketServer.TCPServer((HOST, PORT), MyTCPHandler)` where `MyTCPHandler` is the listener class you implemented.

Server Sockets

```
import SocketServer

class MyTCPHandler(SocketServer.StreamRequestHandler ):
    """
    The RequestHandler class for our server.

    It is instantiated once per connection to the server, and must
    override the handle() method to implement communication to the
    client.
    """

    def handle(self):
        # self.rfile is a file-like object created by the handler;
        # we can now use e.g. readline() instead of raw recv() calls
        self.data = self.rfile.readline().strip()
        print "%s wrote:" % self.client_address[0]
        print self.data
        # Likewise, self.wfile is a file-like object used to write back
        # to the client
        self.wfile.write(self.data.upper())
```

Server Sockets

```
if __name__ == "__main__":
    HOST, PORT = "localhost", 9999

    # Create the server, binding to localhost on port 9999
    server = SocketServer.TCPServer((HOST, PORT), MyTCPHandler)

    # Activate the server; this will keep running until you
    # interrupt the program with Ctrl-C
    server.serve_forever()
```

You can try running in one window:

```
python server.py
```

And in another window

```
bash-4.1$ telnet sslab03 9999
Trying 128.10.25.103...
Connected to sslab03.cs.purdue.edu.
Escape character is '^]'.
Hi how are you?
HI HOW ARE YOU?
Connection to sslab03.cs.purdue.edu closed by foreign host.
bash-4.1$
```

Client Sockets

- To connect to a server, create a socket, then connect to the server and send data.
- Then call receive to get the answer.

Client Sockets

```
import socket
import sys

HOST, PORT = "localhost", 9999
data = " ".join(sys.argv[1:])

# Create a socket (SOCK_STREAM means a TCP socket)
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect to server and send data
sock.connect((HOST, PORT))
sock.send(data + "\n")

# Receive data from the server and shut down
received = sock.recv(1024)
sock.close()

print "Sent:     %s" % data
print "Received: %s" % received
```

Sending mail

- To send e-mail use the SMTMP module

```
# Import smtplib for the actual sending
  function
import smtplib

# Import the email modules we'll need
from email.mime.text import MIMEText

# Create a text/plain message
body="Good luck in the project!"
msg = MIMEText(body)
```

Sending mail

```
fromaddr="grr@cs"
toaddr="grr@cs"

# me == the sender's email address
# you == the recipient's email address
msg['Subject'] = 'Hi CS390...'
msg['From'] = fromaddr
msg['To'] = toaddr

# Send the message via our own SMTP server, but don't
  include the
# envelope header.
s = smtplib.SMTP("localhost")
s.set_debuglevel(1)
s.sendmail(fromaddr, [toaddr], msg.as_string())
s.quit()
```

Multiple Threads

- Multiple threads allow running tasks in parallel with the program.
- To execute a thread use the command define a class that inherits from threading.Thread
- Inside the method run() define the body of the function that will run in the thread.
- Use join to wait until the thread ends.

Threads Example: Unzipping a File in the Background.

```
import threading, zipfile

class AsyncZip(threading.Thread):
    def __init__(self, infile, outfile):
        threading.Thread.__init__(self)
        self.infile = infile
        self.outfile = outfile
    def run(self):
        f = zipfile.ZipFile(self.outfile, 'w', zipfile.ZIP_DEFLATED)
        f.write(self.infile)
        f.close()
        print 'Finished background zip of: ', self.infile

background = AsyncZip('mydata.txt', 'myarchive.zip')
background.start()
print 'The main program continues to run in foreground.'

background.join()    # Wait for the background task to finish
print 'Main program waited until background was done.'
```

Building GUIs in Python

- There are many GUI libraries in Python
  - Tkinter
  - wxWidgets
  - Qt
  - Gtk+
  - FLTK
  - FOX
  - OpenGL

  And more
- Tkinter comes preinstalled in Python and it is the one we will use here.

Tkinter

- Developped at Sun Labs
- Available in Unix, Windows and MacOS
- To use:

```
import tkinter
```

or

```
from tkinter import *
```

Reference: http://www.pythonware.com/library/tkinter/introduction/index.htm

Hello World in Tkinter

```
# File: hello1.py

from tkinter import *

root = Tk()             # Obtains a window
w = Label(root, text="Hello, world!")
w.pack()          # Creates a label
                  # Adds label to window

root.mainloop() # Wait for window events.
```

A More Complex Example

```
# File: hello2.py

from tkinter import *

class App:

    def __init__(self, master):

        frame = Frame(master)
        frame.pack()

        self.button = Button(frame, text="QUIT", fg="red", command=frame.quit)
        self.button.pack(side=LEFT)

        self.hi_there = Button(frame, text="Hello", command=self.say_hi)
        self.hi_there.pack(side=LEFT)

    def say_hi(self):
        print "hi there, everyone!"

root = Tk()

app = App(root)

root.mainloop()
```

Widget Classes

Button

A simple button, used to execute a command or other operation.

Canvas

Structured graphics. This widget can be used to draw graphs and plots, create graphics editors, and to implement custom widgets.

Checkbutton

Represents a variable that can have two distinct values. Clicking the button toggles between the values.

Entry

A text entry field.

Frame

A container widget. The frame can have a border and a background, and is used to group other widgets when creating an application or dialog layout.

Label

Displays a text or an image.

Widget Classes

Listbox

Displays a list of alternatives. The listbox can be configured to get radiobutton or checklist behavior.

Menu

A menu pane. Used to implement pulldown and popup menus.

Menubutton

A menubutton. Used to implement pulldown menus.

Message

Display a text. Similar to the label widget, but can automatically wrap text to a given width or aspect ratio.

Widget Classes

Radiobutton

Represents one value of a variable that can have one of many values. Clicking the button sets the variable to that value, and clears all other radiobuttons associated with the same variable.

Scale

Allows you to set a numerical value by dragging a "slider".

Scrollbar

Standard scrollbars for use with canvas, entry, listbox, and text widgets.

Text

Formatted text display. Allows you to display and edit text with various styles and attributes. Also supports embedded images and windows.

Toplevel

A container widget displayed as a separate, top-level window.

Mixins

- Similar to Java Swing Layouts

- Grid Manager

  The grid geometry manager allows you to create table-like layouts, by organizing the widgets in a 2-dimensional grid. To use this geometry manager, use the grid method.

- Pack Manager

  The pack geometry manager lets you create a layout by "packing" the widgets into a parent widget, by treating them as rectangular blocks placed in a frame. To use this geometry manager for a widget, use the pack method on that widget to set things up.

- Place Manager

  The place geometry manager lets you explicitly place a widget in a given position. To use this geometry manager, use the place method.

Creating a small menu

```
# File: menu1.py

from tkinter import *

def callback():
    print ("called the callback!“)

root = Tk()

# create a menu
menu = Menu(root)
root.config(menu=menu)

filemenu = Menu(menu)
menu.add_cascade(label="File", menu=filemenu)
filemenu.add_command(label="New", command=callback)
filemenu.add_command(label="Open...", command=callback)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=callback)

helpmenu = Menu(menu)
menu.add_cascade(label="Help", menu=helpmenu)
helpmenu.add_command(label="About...", command=callback)

mainloop()
```

Standard Dialogues

```
try:
    fp = open(filename)
except:
    tkMessageBox.showwarning(
        "Open file",
        "Cannot open this file\n(%s)" % filename
    )
    return
```

Dialog Windows

```
# File: dialog1.py

from tkinter import *

class MyDialog:

    def __init__(self, parent):

        top = self.top = Toplevel(parent)

        Label(top, text="Value").pack()

        self.e = Entry(top)
        self.e.pack(padx=5)

        b = Button(top, text="OK", command=self.ok)
        b.pack(pady=5)

    def ok(self):
        print ("value is", self.e.get())
        self.top.destroy()

root = Tk()
Button(root, text="Hello!").pack()
root.update()

d = MyDialog(root)

root.wait_window(d.top)
```

Using Canvas

```
import tkinter
import tkinter.messagebox

top = tkinter.Tk()

C = tkinter.Canvas(top, bg="blue", height=250, width=300)

coord = 10, 50, 240, 210
arc = C.create_arc(coord, start=0, extent=150, fill="red")

C.pack()
top.mainloop()
```

Another Canvas Example.

```
# canvas2.py

from tkinter import *

master = Tk()

w = Canvas(master, width=200, height=100)
w.pack()

w.create_line(0, 0, 200, 100)
w.create_line(0, 100, 200, 0, fill="red", dash=(4, 4))

w.create_rectangle(50, 25, 150, 75, fill="blue")

mainloop()
```

See http://effbot.org/tkinterbook/canvas.htm for more info.