Lecture Notes 8
Introduction to CGI Programming using Python

Anand Tripathi

CSci 4131

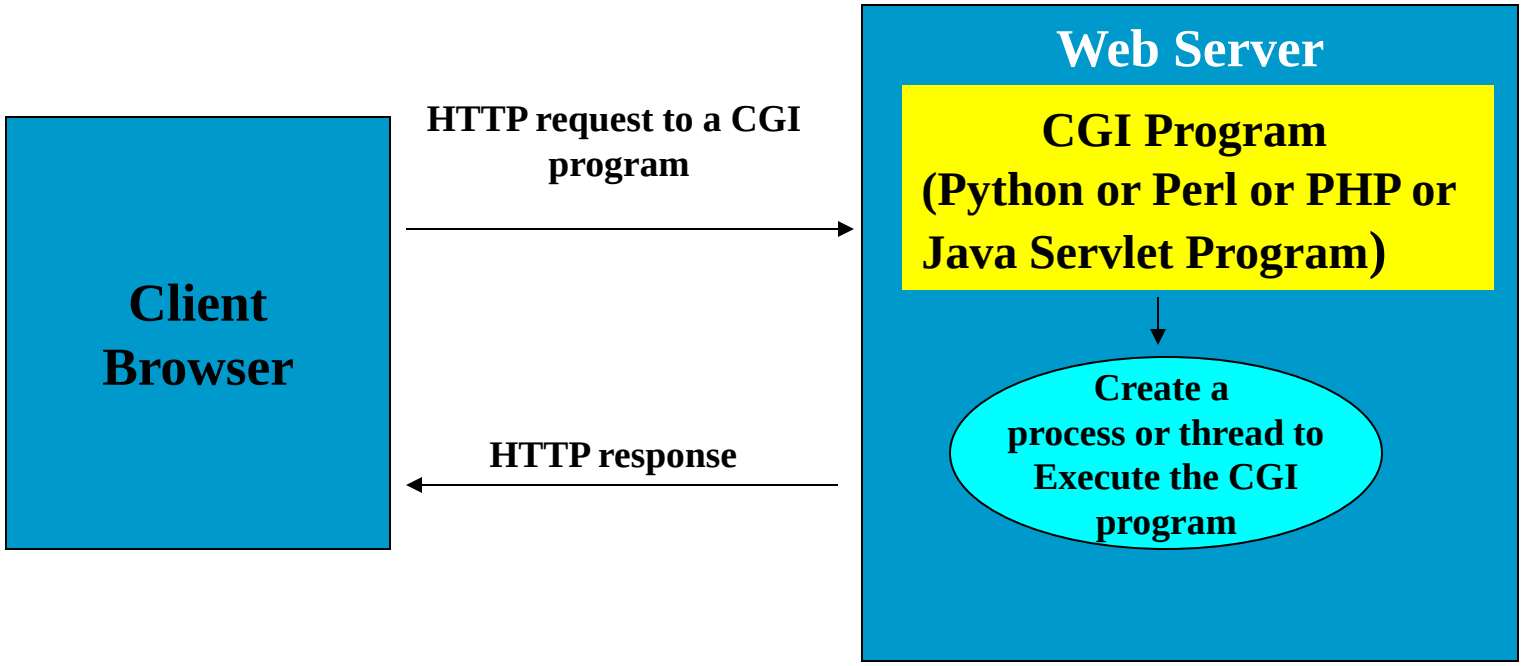
Internet Programming

Server Side Programming

- We will now study techniques used for programming services and applications on a web server.
- On a web server, a hosted application program can perform the following kinds of functions:
  - Dynamically generate web document contents in response to a user request
  - Process the data submitted a web client (browser)
    - Typically data is submitted as part of some form submission
  - Interface with backend database systems or file system to store/retrieve data

Common Gateway Interface (CGI) Protocol

- **We will write programs that execute on the server to handle user requests such as form processing**



Common Gateway Interface

- It is web server interface that allows a client request for specific web resource (URL) to result in execution of a program on the web server rather than sending some stored data to the client.
- The output data generated by the executed program is sent to the client.
- CGI interface provides mechanisms for passing arguments or input data to the CGI program based on the client request.
  - For example, Query_String on a for submission is passed as either an environment variable or as standard input

Common Gateway Interface

- CGI is language independent
- Many languages provide library functions to support CGI programming.
  - Perl, Python, PHP, Ruby, Java Servlets, JSP,……

- In this course we will learn Python and PHP for CGI programming.

Topics

- Part I: Introduction to the Python Programming Language

- Part 2: CGI Programming and Form Processing using Python

Part I:
Introduction to Python

Features

- High level language
  - No need to worry about low level details like memory management etc.
- Interpreted
  - Compilation to binary not needed, run directly from source code. Internally converted to bytecode and then to native language and run it.
- High level data types
  - flexible arrays, dictionaries etc.
- Dynamic typing
  - type checks are performed mostly at run time.
  - bind a name to objects of different types during program execution.
- Support both procedure oriented and object oriented programming.

Using Python Interpreter

- Generally invoked by typing command “python” in shell.

```
$ python

Python 2.7 (#1, Feb 28 2010, 00:02:06)
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

Data Structures in Python

Data Structures in Python

- Lists (arrays)
  - A list can store items of different types
  - Items can be accessed using index
- Dictionary (hashes(
  - Key-Value pairs
- Tuples
  - Ordered set of immutable items

Lists

- Lists are used for storing data items of any type
- Items can be added, removed, accessed using integer index starting with 0.
- List can be sorted or reversed
- A slice of the list can be obtained for a specified index range.

Lists

```
>>> a =['apple', 'banana', 'cherry']
>>> a
['apple', 'banana', 'cherry']
>>> a[0]
'apple'
>>> a[2]
'cherry'
>>> a[-1]
'cherry'
>>> a[-2]
'banana'
>>> a = a + ['bread', 'milk']
>>> a
['apple', 'banana', 'cherry', 'bread', 'milk']
>>> a[2] = 20
>>> a
['apple', 'banana', 20, 'bread', 'milk']
```

Lists

```
>>> a.sort()
>>> a
[20, 'apple', 'banana', 'bread', 'milk']
>>> a[2:4]
['banana', 'bread']
>>> a[:2]
[20, 'apple']
>>> a.reverse()
>>> a
['milk', 'bread', 'banana', 'apple', 20]
>>>
```

Lists

Some list methods are as follows:

len(list)
returns the length of list.

list.append(x)
Add an item to the end of the list; equivalent to a[len(a):] = [x].

list.insert(i, x)
Insert an item at a given position. The first argument is the index of the element before which to insert

list.remove(x)
Remove the first item from the list whose value is x. It is an error if there is no such item.

list.pop([i])
Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the last item in the list. Parameter is optional.

list.index(x)
Return the index in the list of the first item whose value is x. It is an error if there is no such item.

list.count(x)
Return the number of times x appears in the list.

list.sort()
Sort the items of the list, in place.

list.reverse()
Reverse the elements of the list, in place.

Lists

- **Some list methods are illustrated as below.**

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print a.count(333), a.count(66.25), a.count('x')
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

Dictionaries

- An unordered set of *key: value* pairs, with the requirement that the keys are unique (within one dictionary).

- A pair of braces creates an empty dictionary: {}

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
```

Dictionaries

- **Possible to delete a *key:value* pair with del.**

```
>>> del tel['sape']
```

- **Get all keys in dictionary as list using keys() method.**

```
>>> tel['irv'] = 4127

>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}

>>> tel.keys()
['guido', 'irv', 'jack']
```

- **Check for membership of key in dictionary using 'in' operator.**

```
>>> 'guido' in tel
True
```

Lists

Using Lists as Stacks.

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

Tuples

- Similar to list but immutable. (You cannot modify a tuple.)
- Consists of a set of data items, which can be of different types.
- Tuples are always enclosed in parentheses, with a list of items separated by command

  t = (12345, 54321, 'hello!‘)

Tuples

```
>>> t = (12345, 54321, 'hello!‘)
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Lists

Sets

- Unordered collection with no duplicate elements.
- Basic uses include membership testing and eliminating duplicate entries.
- Support mathematical operations like union, intersection, difference.
- set() function can be used to create sets.

Sets

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> fruit = set(basket)              # create a set without duplicates
>>> fruit
set(['orange', 'pear', 'apple', 'banana'])


>>> 'orange' in fruit                # fast membership testing
True
>>> 'crabgrass' in fruit
False
```

Sets

```
>>> # Demonstrate set operations on unique letters from two words
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                                  # unique letters in a
set(['a', 'r', 'b', 'c', 'd'])
>>> a - b                              # letters in a but not in b
set(['r', 'd', 'b'])
>>> a | b                              # letters in either a or b, i.e. a union b
set(['a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'])
>>> a & b                              # letters in both a and b, i.e. a intersection b
set(['a', 'c'])
>>> a ^ b                              # letters in a or b but not both, (a union b) - (a
 intersection b)
set(['r', 'd', 'b', 'm', 'z', 'l'])
```

String

- **Can be enclosed in single quotes or double quotes.**

'spam eggs’ ,
‘doesn\’ t’ ,
“doesn't",
‘ "Yes," he said.’ ,
“ \"Yes,\" he said. ”,
‘ "Isn\'t," she said.’

- **Continuation lines can be used, with backslash as last character.**

hello = "This is a rather long string containing\n\
several lines of text just as you would do in C.\n\
    Note that whitespace at the beginning of the line is\
 significant.”

String

A long string with any special characters, spaces, new lines, can be surrounded with pair of matching triple-quotes: ””” or ‘’’.

- **End of lines don't need to be escaped but they will be included in string. This will display the blob of text as it is inside the triple quote**.

```
print """
Usage: thingy [OPTIONS]
     -h                      Display this usage message
     -H hostname       Hostname to connect to
"""
```

String

- String concatenation and repeat

```
>>> word = 'Help' + 'A'
>>> word
'HelpA'
>>> '<' + word*5 + '>'
 ‘<HelpAHelpAHelpAHelpAHelpA>'
```

- Strings can be indexed, the first character of a string has subscript (index) 0.

```
>>> word[4]
'A'
>>> word[0:2]
'He'
>>> word[2:4]
 ‘lp'
>>> word[:2]    # The first two characters
'He'
>>> word[2:]    # Everything except the first two characters
'lpA'
```

String

- **String comparison**

```
>>> word = ‘hello’
>>> if word == ‘hello’ :
        print ‘Strings are equal’
‘Strings are equal’
```

- **Split string into list of words based on a delimiter**.

```
word = "Hello World"
>>> word.split(' ')  # Split on whitespace
['Hello', 'World']
```

- **Join list of words to form a string**

```
>>> words = [ ‘hello’ ,’ world’ ]
>>> print ‘ ‘.join(words)  # add a blank between every word
hello world
```

String Manipulations

- **Strip white-spaces from ends of string.**

```
strip()     #removes from both ends
lstrip()    #removes leading characters (Left-strip)
rstrip()    #removes trailing characters (Right-strip)

>>> word = "    xyz    "
>>> print word
    xyz
>>> print word.strip()
xyz
>>> print word.lstrip()
xyz
>>> print word.rstrip()
    xyz
```

Pattern Matching

- **The re module provides Perl-style regular expression patterns.**
- **re.compile(‘pattern’) is used to create pattern object.**

```
import re    //  You must import the re module
p = re.compile(‘[a-z]+’)
```

- **match(‘string’) return match object and None in case there is no match.**
- **Using pattern object.**

```
m = p.match( 'string goes here' )
        // check if pattern matches at beginning of the string
if m:
   print 'Match found: ', m.group()
else:
   print 'No match’
```

m.group() above return the string matched by pattern.

Pattern matching characters

Character	matches
^	beginning of a string
$	end of string
.	Any character except newline
+	One of more occurrences of a pattern
*	Zero ore more occurrences of a pattern
\b	Word boundary
\w	any alpha-numeric character
\W	any non alpha-numeric character
[a-z]	any character in a through z

Example

Notice that “r” means raw text data
And special meaning of \ is ignored.

```
>>> expr = re.compile( r“\bd\w*d\b" )
>>> result = expr.findall( "We decided it was a dead matter." )
>>> print len( result )
2
>>> for item in result:
...     print  item
```

- What will be the output?

Control statements & functions

Conditional Statements

- **'else' is optional.**
- **'elif' is short for 'else if'**

```
>>> x = int ( raw_input("Please enter an integer: ")  )
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print 'Negative changed to zero'
... elif x == 0:
...     print 'Zero'
... elif x == 1:
...     print 'Single'
... else:
...     print 'More'
...
More
```

Iteration using “for loop”

- **Iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.**

```
>>> words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print w, len(w)
...
cat 3
window 6
defenestrate 12
```

- **range() comes handy to iterate over sequence of numbers.**

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(5, 10)
[5, 6, 7, 8, 9]
>>> range(0, 10, 3)   // starting with 0,numbers upto 10, with strides of 3
[0, 3, 6, 9]
```

Iteration using “for loop”

```
>>> a = ['Mary', 'had', 'a', 'little', 'lamb']
>>> for i in range(len(a)):
...     print i, a[i]
...
0 Mary
1 had
2 a
3 little
4 lamb
```

break statement in “for loop”

- **The ‘break’ statement breaks out of the smallest enclosing <u>for</u> or <u>while</u> loop.**

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print n, 'equals', x, '*', n/x
...             break
...         else:
...                     If x==(n-1):
...                             print n, 'is a prime number'
...
```

continue statement in “for loop”

- **The ‘continue’ statement, also borrowed from C, continues with the next iteration of the loop.**

```
>>> for num in range(2, 10):
...     if num % 2 == 0:
...         print "Found an even number", num
...         continue
...     print "Found a number", num
Found an even number 2
Found a number 3
Found an even number 4
Found a number 5
Found an even number 6
Found a number 7
Found an even number 8
Found a number 9
```

Iteration on Dictionaries and Sets

- **Iteration on dictionary keys**

```
>>> metals ={'gold':1000,'silver':200,'copper':50}
>>> metals
{'copper': 50, 'silver': 200, 'gold': 1000}
>>> for key,value in metals.iteritems():
...     print key,value
copper 50
silver 200
gold 1000
```

- **Iteration on a set**

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for f in set(basket):
...     print f
...
orange
pear
apple
banana
```

Functions

- Keyword *def* is used to define function.
- Followed by function name and parenthesized list of parameters.
- Statements in function body start from next and are indented.

Functions

- **Fibonacci example**

```
>>> def fib(n):    # write Fibonacci series up to n
...     """Print a Fibonacci series up to n."""
...     a, b = 0, 1
...     while a < n:
...         print a,
...         a, b = b, a+b
...
>>> # Now call the function we just defined:
... fib(2000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597
```

Functions

- **Default arguments**

```
def ask_ok   (prompt, retries=4, complaint='Yes or no, please!'):
    while True:
        ok = raw_input(prompt)
        if ok in ('y', 'ye', 'yes'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries < 0:
            raise IOError('refusenik user')
        print complaint
```

Functions

- raw_input() reads a line from input and return string by stripping trailing newline.
- Mandatory argument call: ask_ok('Do you really want to quit?’)
- Optional argument calls:
  - ask_ok('OK to overwrite the file?', 2)
  - ask_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')

Function Arguments

- Two ways parameters are passed:
  - By value (any modification to it by the function code is only visible in the function and not outside the function)
  - By reference

- Scalar arguments (such as string or number) are passed by value
- For list and dictionary argument, a reference (pointer) to the argument is passed to the function
- Take care if data types are mutable or immutable.

Function Arguments

- Example for List - a mutable type

```
def try_to_change_list_contents(the_list):
    print 'got', the_list
    the_list.append('four')
    print 'changed to', the_list

outer_list = ['one', 'two', 'three']
print 'before, outer_list =', outer_list
try_to_change_list_contents(outer_list)
print 'after, outer_list =', outer_list
```

Output:
before, outer_list = ['one', 'two', 'three']
got ['one', 'two', 'three']
changed to ['one', 'two', 'three', 'four']
What will be printed here?

Function Arguments

- Example for List - a mutable type

```
def try_to_change_list_contents(the_list):
    print 'got', the_list
    the_list.append('four')
    print 'changed to', the_list

outer_list = ['one', 'two', 'three']
print 'before, outer_list =', outer_list
try_to_change_list_contents(outer_list)
print 'after, outer_list =', outer_list
```

Output:

```
before, outer_list = ['one', 'two', 'three']
got ['one', 'two', 'three']
changed to ['one', 'two', 'three', 'four']
after, outer_list = ['one', 'two', 'three', 'four']
```

Functions Arguments

- **Example for List - a immutable type**

```
def try_to_change_list_reference(the_list):
    print 'got', the_list
    the_list = ['and', 'we', 'can', 'not', 'lie']
    print 'set to', the_list

outer_list = ['we', 'like', 'proper', 'English']
print 'before, outer_list =', outer_list
try_to_change_list_reference(outer_list)
print 'after, outer_list =', outer_list
```

Output:
before, outer_list = ['we', 'like', 'proper', 'English']
got ['we', 'like', 'proper', 'English']
set to ['and', 'we', 'can', 'not', 'lie']
What will be printed here?

Functions Arguments

- **Example for List - a immutable type**

```
def try_to_change_list_reference(the_list):
    print 'got', the_list
    the_list = ['and', 'we', 'can', 'not', 'lie']
    print 'set to', the_list

outer_list = ['we', 'like', 'proper', 'English']
print 'before, outer_list =', outer_list
try_to_change_list_reference(outer_list)
print 'after, outer_list =', outer_list
```

Output:

```
before, outer_list = ['we', 'like', 'proper', 'English']
got ['we', 'like', 'proper', 'English']
set to ['and', 'we', 'can', 'not', 'lie']
after, outer_list = ['we', 'like', 'proper', 'English']
```

Functions Arguments

- **Example for String - an immutable type**

```
def try_to_change_string_reference(the_string):
    print 'got', the_string
    the_string = 'In a kingdom by the sea'
    print 'set to', the_string

outer_string = 'It was many and many a year ago'
print 'before, outer_string =', outer_string
try_to_change_string_reference(outer_string)
print 'after, outer_string =', outer_string
```

Output:

```
before, outer_string = It was many and many a year ago
got It was many and many a year ago
set to In a kingdom by the sea
after, outer_string = It was many and many a year ago
```

Modules

- Easier to maintain code bigger script by splitting into multiple scripts.
- Module is a file containing Python definitions and statements.
- File name is the module name with suffix ‘.py’.
- Use ‘import’ keyword to import the module.
- Use ‘from’ keyword to import specific function or definition from module.
- Within a module the global variable ‘__name__’ contains module name.

Modules

```
# Fibonacci numbers module, saved in a file as fibo.py
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b
def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Modules

```
On Interpreter:
>>> import fibo
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'

#importing specific module
>>> from fibo import fib, fib2
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

Modules

- **To execute module as script add following at end of script.**

```
#add following at end of script
if __name__ == "__main__":
    func()
```

- When you run a script global variable for module name '__name__' is set to "__main__"
- You can run from the command line as: python <scriptname.py>

Modules

```
# Fibonacci numbers module, saved in a file as fibo.py
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n): # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

#add following at end of fibo.py
if __name__ == "__main__":
    fib(50)
```

```
$ python fibo.py
1 1 2 3 5 8 13 21 34
```

File I/O

- Use open(filename, mode) to return a file object for I/O operations
- First argument is string containing file name.
- Mode describes the way in which file will be used.
  - ‘r’ - for only reading
  - ‘w’ - for only writing
  - ‘a’ - for appending data to end
  - ‘r+’ - for both reading and writing

```
>>> f = open('workfile', 'r')
```

File I/O

Use read(‘size’) on file object to read data of length at most ‘size’ and return it as string.

If ‘size’ is omitted then entire contents of file will be read and returned.

```
>>> f.read()
'This is the entire file.\n'
>>> f.read()
''
```

Use readline() to read a single line from file.

Newline character (‘\n’) is left at the end of string

Newline omitted on last line of file , if file doesnt end in newline.

Returns empty string if end of file is reached. Blank line is returned as ‘\n’.

```
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'Second line of the file\n'
>>> f.readline()
''
```

File I/O

- Looping over file object to read lines is memory efficient, fast and simple.

```
>>> for line in f:
        print line,

This is the first line of the file.
Second line of the file
```

- If open in writing mode then use write(string), to write the contents of string to file.

```
>>> f.write('This is a test\n')
```

- Close the file when you are done with it to free up any system resources taken by by open file.

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

File I/O

- **Copy file from source to destination.**

```
#open source file for reading
fromFile = open('sourceFile.txt', ‘r')

#open destination file for writing
toFile = open('destinationFile.txt', 'w')

#iterate over the lines of source
for line in fromFile:
    #write line from source file to destination file
    toFile.write(line)

fromFile.close()
toFile.close()
```

Exceptions

- Syntactically correct statements may cause an error when executed.

```
>>> 10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
ZeroDivisionError: integer division or modulo by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: cannot concatenate 'str' and 'int' objects
```

Exceptions

- Possible to write program that handle selected exceptions.
- Use of ‘try’ and ‘except’ clause.
- Statement between ‘try’ and ‘except’ keyword is executed.
- If no exception then ‘except’ clause is skipped and execution of the is finished.
- If an exception occurs during ‘try’ clause execution, rest of clause is skipped.

  If it matches the exception named after ‘except’ keyword, except clause is executed and execution continues after ‘try’ statement.
- If exception don’t match the exception in except clause, it is passed to outer ‘try’ blocks. If no handler is found, it’s an unhandled exception and execution stops with error.

Exceptions

```
import sys

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

Part 2:
CGI Programming using Python

See Python Documentation Website

Python

- Python modules like cgi, cookie, smtplib, urllib, ftplib and others provide extensions that can be use to write CGI scripts for almost any task.
- Save your Python program in your .www directory with extension .cgi on CSELab server.
- Basic CGI program.

```
#!/usr/bin/python

print "content-type: text/html\n\n“
print “Hello Snow !”
```

- Notice extra blank line after printing content-type header.
- See this example

Python

See this example

```
#!/soft/python-2.5-bin/python

print "Content-type: text/html"
print
print "<!DOCTYPE html>"
print "<html>"
print "<head>"
print "<title>Python CGI Example</title>"
print "</head>"
print "<body>"
print "<p>Hello, world!</p>"
print "</body>"
print "</html>"
```

Python

- See this example
- CGI Program to display OS environment variables
- This is similar to the “echo.cgi” used earlier in forms.

```
#!/usr/bin/python

import os
print “Content-type: text/html\n\n"
print "hello snow!"
print "<br/>  <font size=+1>  Environment  </font>  <br/>"
for param in os.environ.keys():
    print "<b>%20s</b>: %s<br>" % (param, os.environ[param])
```

Python

- CGI program displaying HTML.
- Note use of triple quote.
- See this example

```
#!/usr/bin/python

HTML_TEMPLATE = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//EN">
<html><head><title>Hello World</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1">
</head><body><h1>Hello World</h1>
</body>
</html>"""

print "content-type: text/html\n\n"
print HTML_TEMPLATE
```

Form processing

- Simple form processing example
- The following html file will present a form which on submission will be handled by Python program HelloHandle.cgi
- See this program

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <title>Form</title>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
  <body><h1>Enter your details</h1>
    <form action=“helloHandle.cgi" method="POST" enctype="multipart/form-data">
       First name: <input name="firstname" type="text"></br>
       Last name: <input name="lastname" type="text">
      <input name="submit" type="submit">
    </form>
  </body>
</html>
```

Form processing

- FieldStorage() method from module cgi can be used to retrieve request parameters in a form.

```
#!/usr/bin/python
#helloHandle.cgi
import cgi

print 'content-type: text/html\n\n'

form = cgi.FieldStorage()

#get the value of ‘firstname’ input
firstName = form['firstname'].value

#get the value of ‘lastname’ input
lastName = form['lastname'].value

print 'Hello ' + firstName + ' ' + lastName
```

Form processing

- **Form display using CGI**
- See this example

```
#!/usr/bin/python
HTML_TEMPLATE = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Form</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body><h1>Enter your details</h1>
    <form action="FormHandler.cgi" method="POST" enctype="multipart/form-data">
        First name: <input name="firstname" type="text"></br>
        Last name: <input name="lastname" type="text">
        <input name="submit" type="submit">
    </form>
</body>
</html>"""

print "content-type: text/html\n"
print HTML_TEMPLATE
```

Form processing

- FieldStorage() method from module cgi can be used to retrieve request parameters in a form.
- Use of formatted output string using dictionary

```
#!/usr/bin/python
#content of FormHandler.cgi
import cgi
# The following module is for debugging purpose to display error messages
import cgitb; cgitb.enable()

HTML_TEMPLATE = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
<head>
    <title>Form Details</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
    <h1>%(MESSAGE)s</h1>
</body>
</html>"""
```

Form processing

- FieldStorage() method from module cgi can be used to retrieve request parameters in a form.
- Use of formatted output string using dictionary

```
def parseForm ():
    form = cgi.FieldStorage()
    firstName = form['firstname'].value
    lastName = form['lastname'].value
    print HTML_TEMPLATE % {'MESSAGE' : 'Hello ' + firstName + ' ' + lastName}

print 'content-type: text/html\n'
parseForm()
```

CGI Module

- import cgi

You should also import cgitb -- traceback for debugging

- import cgitb
- cgitb.enable()

Another way:

- cgitb.enable(display=0, logdir="/home/userHome/logdir")

FieldStorage Class…(1)

- FieldStorage class is used for accessing file data.
  - Usage example:

    formData = cgi.FieldStorage()
- FieldStorage can be used like a dictionary
  - Exmple
    - formData[“fieldName”]
    - formData[“firstName”]
- The object for a field-name stored is itself a FieldStorage
- formData[“fieldName”].value returns the string value for the field
  - Another way: use getvalue() method
- If multiple values are associated with a field-name then getvalue() function returns a list

FieldStorage Class…(2)

- You can obtain the list of all keys in FieldStorage
  - Example

    formData.keys()

- One can use “in” operator to check if a field-name is present in the FieldStorage
  - Example

    if “firstName” not in formData:

    print “Error in submision: Please enter firstname”

FieldStorage Class…(3)

- Suppose that a field-name represents an uploaded file
- File is accessed as follows:

  Example:

  fileitem = formData[“fileFieldName”]
- fileitem object will have several attributes:
  - value
  - file
  - filename
- fileitem.value will return the entire file content
- fileitem.getvalue() will return the entire file content

FieldStorage Class…(4)

- “file” property of fileitem provides file-based abstraction for accessing uploaded data and has several methods for reading data

```
while 1:
    line = fileitem.file.readline()
    If not line: break
```

- You can read a specified number of bytes from the file:

```
chunk = fileitem.file.read(4096)
```

Form processing

- **Form containing multiple selections.**
- **See this example**

```
<html>
<body>
  <form method="GET" action="multiSelFormHandle.py" >
   Name: <input type="text" name="firstname"  value="" size="20" />
   Lastname: <input type="text" name="lastname"  value=""  size = "20" />
   Age: <input type="text" NAME="Age"  />
  Please indicate your favorite topics in computer science: <br/>
   <select name="subjects"  multiple>
       <option  value="os"> Operating System </option>
       <option  value="internet"> Internet Programming </option>
       <option  value="compls"> Compilers </option>
       <option  value="arch"> Architecture </option>
       <option  value="theory"> Theory of Computation </option>
       <option  value="numbs"> Numerical Analysis </option>
       <option  value="graphs"> Graphics </option>
       <option  value="parallel"> Parallel Programming </option>
       <option  value="dbms"> Databases </option>
       <option  value="oo"> Object-Oriented Programming </option>
   </select>
   <input type="submit" value="SUBMIT form" />
   <input type="reset" value="CLEAR form TO START AGAIN" />
 </form>
</body>
</html>
```

Form processing

- **CGI script to handle multiple selections.**

```
#!/soft/python-2.5-bin/python
import cgi;

print "Content-type: text/plain\n\n";
form = cgi.FieldStorage();
print "Hello " +  form['firstname'].value + ","
print "You are " + form['Age'].value + " years old."
choices = form.getvalue("subjects","");
for item in choices:
    if (item == 'os'):
       print "You like operating systems"
    elif (item == 'internet'):
       print "You like Internet Programming"
    elif (item == 'arch'):
       print "You like Computer Architecture"
    elif (item == 'compls'):
       print "You like Compilers"

print  "You like ", len(choices),    " subjects"
print "Thank you for submitting the form. "
```

Form processing

•Handle both request and response in same file.

```
#!/usr/bin/python
#helloSingleFile.py
import cgi
import cgitb; cgitb.enable()
import os

#html form string
HTML_FORM_TEMPLATE = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
<head>
<title>Form</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body><h1>Enter your details</h1>
<form action="helloHandle.py" method="POST" enctype="multipart/form-data">
    First name: <input name="firstname" type="text"></br>
    Last name: <input name="lastname" type="text">
    <input name="submit" type="submit">
</form>
</body>
</html>"""
```

Form processing

```
#html response to form
HTML_RESP_TEMPLATE = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
    <head>
        <title>Form Details</title>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    </head>
<body>
    <h1>%(MESSAGE)s</h1>
</body>
</html>"""
def parseForm ():
   form = cgi.FieldStorage()
   firstName = form['firstname'].value
   lastName = form['lastname'].value
   print HTML_RESP_TEMPLATE % {'MESSAGE' : 'Hello ' + firstName + ' ' + lastName}
print 'content-type: text/html\n\n'
form = cgi.FieldStorage()
if 'firstname' in form and len(form['firstname'].value) > 0:
   parseForm()
else:
   print HTML_FORM_TEMPLATE
```

File upload in form processing

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
    <title>File Upload</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
    <h1>File Upload</h1>
    <form action="uploadHandle.cgi" method="POST" enctype="multipart/form-
data">
        File name: <input name="file_input" type="file">
        <br>
        <input name="submit" type="submit">
    </form>
</body>
</html>
```

Form processing

File upload form handler (uploadHandle.cgi).
See Example link on course webpage

See this example

```
#!/usr/bin/python
import cgi
import cgitb; cgitb.enable()
import os

UPLOAD_DIR = "/home/someUser/coursework/iprog/tmp"
HTML_TEMPLATE = """<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
    <head>
        <title>File Upload</title>
        <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
    </head>
<body>
    <h1>File Upload</h1>
    <h1>%(MESSAGE)s</h1>
</body>
</html>"""
```

Form processing

```
def save_uploaded_file (form_field, upload_dir):
    form = cgi.FieldStorage()
    if not form.has_key(form_field):
        print HTML_TEMPLATE % {'MESSAGE':"Error: file item not in form"}
        return
    fileitem = form[form_field]
    if not fileitem.file or len(fileitem.filename) ==0:
        print HTML_TEMPLATE % {'MESSAGE': "Error: file not found"}
        return
    fout = file (os.path.join(upload_dir, fileitem.filename), 'w')
    while 1:
        chunk = fileitem.file.read(100000)
        if not chunk: break
        fout.write (chunk)
    fout.close()
    print HTML_TEMPLATE % {'MESSAGE':'File uploaded successfully in ' + upload_dir}

print 'content-type: text/html\n\n'
save_uploaded_file ("file_input", UPLOAD_DIR)
```

How to generate thumbnail

```
#create thumbnail of 140x140
    import  Image
    size = (140, 140)
    im=Image.open('filename.jpg')
    im.thumbnail(size)
    im.save('filename_tn.jpg', "JPEG")
```

How to redirect to another URL

```
<html>
<head>
  <meta http-equiv="refresh" content="5;http://www.cs.umn.edu" />
</head>
<body>
   <h2> This page has moved to  http://www.cs.umn.edu
  </h2>
</body>
</html>
```

See this example

How to redirect to another URL using Python CGI Program

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

#print header
print "Content-type: text/html"
print "Location: http://www.cs.umn.edu“
print
print "<!DOCTYPE html>"
print "<html>"
print "<head>  </head>"
print "<body>  </body>"
print "</html>"
```

See this example