Assignment 1 Overview

CSCI 4140 Tutorial 3
Feb. 4
Qin Chuan

Assignment 1

- Web Instagram
  - Upload an image
  - Apply filters on the image
  - Save result in server
  - Generate permanent link

- More functions
  - Undo filters
  - Resume of process
  - Even browser is closed and re-opened

Requirement

- Languages
  - HTML
  - Python

No Javascript is allowed

JavaScript will be blocked during demo

- Permanent Storage
  - Database (MySQL, SQLite, …)
  - Image Storage Directory

Testing and Demo

- OS
  - Mac, Linux / Windows
  - Actually it does not matter …

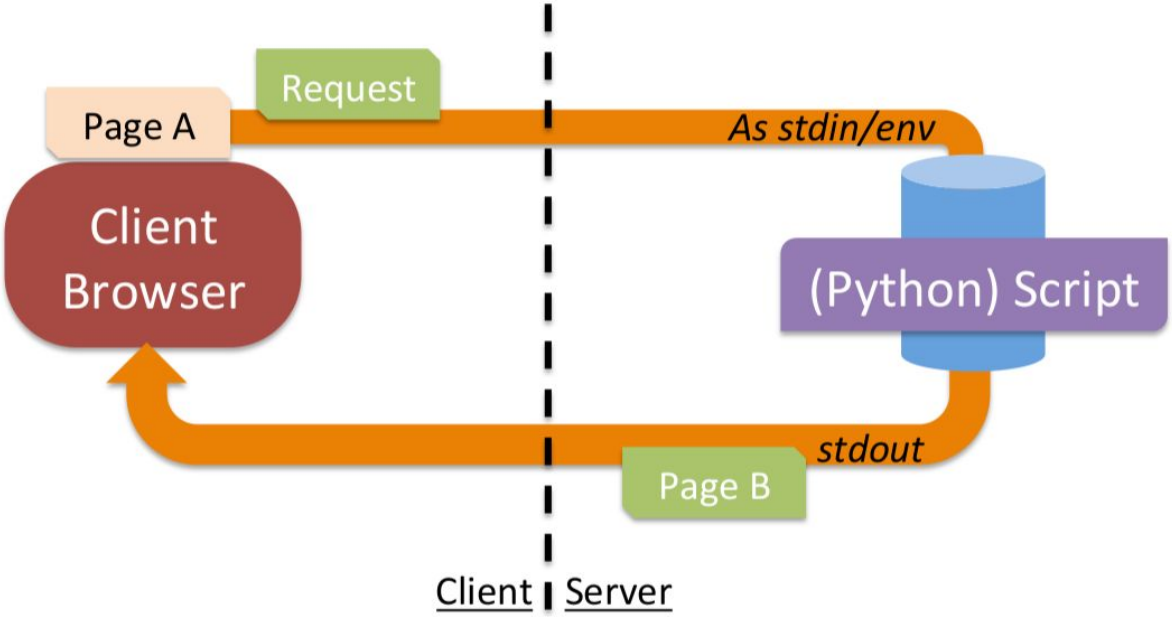
- Browser
  - Mozilla Firefox 4.0 or above; or
  - Google Chrome 32.0 (latest stable ver.)

Recommended Workflow

- Album Display
  - Sort by complete time only
  - Pagination (2x4 images each page)
- Upload Image
  - Save in server
  - Generate URL (Assume no image operations performed)
- Image Operation
  - Apply filters …
  - Undo
- Resume
- Initialize Script
  - Wrap up your work

Dynamically Generating Web Page

- Generating HTML page using script



- Content of page can be controlled by script

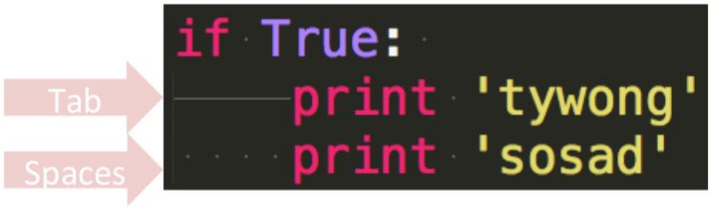
Python

- Scripting language
  - Interpreter No need to compile
  - Code is read when being executed
  - No error raise if the code is not executed (due to branching)

- Duck Typing
  - Variable do not bind with a type / class
  - Depends on value / objected stored
  - Can call any functions from any class
  - Raise exception if objects stored in
  - variable do not implement the function

Python: Code Structure

- Indentation to denote block
  - Use either tab, or spaces as indentation level
    Not mix
  - Raise **IndentationError** if interpreter cannot parse
  - Or maybe unexpected outcome without exception …




```
>>> if True:
...     print 'tywong'
...     print 'sosad'
  File "<stdin>", line 3
    print 'sosad'
                ^
IndentationError: unindent does not match
any outer indentation level
```

  - Turn on ‘show space’ in your editor if needed
- No semicolon
  - Semicolon acts nothing

Python: Debugging Tips

- Debugging maybe painful
  - Check Traceback printed

```
if False:
    a = 1
print a
```

```
>>> if False:
...     a = 1
...
>>> print a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

- To force terminate python script

```
sys.exit(0)
```

  - You will need **sys** module

Exception Handling

- Exception is raised when there is problem when executing your code

```
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

```
>>> 4 + wc
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'wc' is not defined
```

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```

```
>>> while False
  File "<stdin>", line 1
    while False
               ^
SyntaxError: invalid syntax
```

- Or raise an exception in your code

```
>>> raise Exception('Something happened')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Exception: Something happened
```

Exception Handling

- To gracefully handle the exception …

```
try:
    # Do something ...
except KeyError as e:
    print e
    # Handle key error ...
except NameError as e:
    # Handle name error
finally:
    # Do something ...
```

Multiple except block if your need to catch multiple exception type

Optional

- If you just don't want to do anything …

```
try:
    # Do something
except Exception:
    pass
```

Not a good practice!

Modules

- Many build-in modules available
  - E.g. **`sys, math, cgi`**

- Import before use

```
import sys

sys.stdout.write("sosad")
```

```
>>> import sys
>>>
>>> sys.stdout.write("sosad")
sosad>>>
```

- Python Doc: https://docs.python.org/2/contents.html

Modules

- To modularize your code (separate into multiple files)
  - Name your python source file **`<module>.py`**
  - In your cgi (or main python source file), add **`import <module>`**
  - Every time you use functions / variables inside module, add **`<module>.`**

csci4140.py

```
course = 'csci4140'

def foo():
    print 'sosad'
```

main.py

```
import csci4140

csci4140.foo()

print csci4140.course
```

```
>>> import csci4140
>>> csci4140.foo()
sosad
>>> print csci4140.course
csci4140
```

  - **`<module>.pyc`**: bytecode for python's virtual machine

String and String Formatting

- String: Single quote (') or double quote (")
  - Depends on string content (for avoid escaping quote)

```
"Alice's apple"
'My "work"'
```

- String formatting
  - Just like **printf** in C
  - Substitute value into string

```
print "%s%d Tutorial %d" % ('csci', 4140, 2)
```

String with format

Fields

- Output: `CSCI4140 Tutorial 2`

Here Document

- Here-Document
  - Multi-line strings
- Handy when hardcoding long string / HTML code

```
print '''tywong
sosad
csci
4140'''
```

```
>>> print '''tywong
... sosad
... csci
... 4140'''
tywong
sosad
csci
4140
```

- String formatting is also allowed

```
print """csci
4140
%s""" % ('Tutorial')
```

```
>>> print """csci
... 4140
... %s""" % ('Tutorial')
csci
4140
Tutorial
```

Named and Optional Arguments

- Found a lot in Python doc

Argument name

Default argument

```
def area(a, b=0, type='circle'):
    if (type == 'circle'):
        return a * a * 3.1415
    elif (type == 'square'):
        return a * a
    elif (type == 'rectangle'):
        return a * b
```

a = 5, b = 0, type = 'circle'

```
>>> area(5)
78.53750000000001
```

a = 5, b = 0, type = 'square'

```
>>> area(5, type='square')
25
```

a = 5, b = 0, type = 'rectangle'

```
>>> area(5, type='rectangle')
0
```

a = 5, b = 3, type = 'circle'

```
>>> area(5, 3, type='rectangle')
15
```

Using Python for CGI

- Regard as printing response to stdout

helloworld.cgi

Shebang

HTTP Header

Content (HTML)

```
#!/usr/bin/python

print 'Content-Type: text/html'
print

print '<html>'
print '<body><h3>Hello World. </h3></body>'
print '</html>'
```

Run from shell

```
✔ 04:42:43 jimmy@JimmyHMac ~/openshift/tutorial
 master python helloworld.cgi
Content-type: text/html

<html>
<h3><body>Hello World! </body></h3>
</html>
```

Browser

Hello World!

tutorial3/helloworld.html

Retrieving Parameters

- **`cgi`** module
  - Process user input (both GET and POST method)
  - How to use?
  - Import **`cgi`** module

```
import cgi
```

  - Parse the input to **`FieldStorage`**

```
form = cgi.FieldStorage()
```

More detail: https://docs.python.org/2/library/cgi.html

`tutorial3/form_process.html`

Retrieving Parameters

- **`cgi`** module
  - Process user input (both GET and POST method)
  - How to retrieve parameters?
  - Access as dictionary
  - **`getvalue(key)`**: Return a string or list
  - **`getlist(key)`**: Always return a list

Default value *(optional, if parameter not exist / empty)*

```
user = form.getvalue('login', None)
passwd = form.getlist('passwd')[0]
action = form['action'].value
```

If empty input, this key will not exist in dictionary

Get the first element *what if zero length?*

`tutorial3/form_process.html`

File Upload

- File upload using HTTP POST request

- We need (at least) **two pages** to handle upload
  - **Form** accepting user's input and send request
  - Process user's request on server, and generate **result page** (e.g. upload finish confirmation)

HTML Form

upload_form.html

```html
<form enctype="multipart/form-data"
      action="upload.cgi" method="POST">
    Choose an image (.jpg .gif .png): <br />
    <input type="file" name="pic"
           accept="image/gif, image/jpeg, image/png" />
    <br />
    <input type="submit" value="Upload" />
</form>
```

Browser

tutorial3/upload_form.html

HTML Form

Encoding type: `multipart/form-data` for binary data (file)

upload_form.html

```
<form enctype="multipart/form-data"
      action="upload.cgi" method="POST">
    Choose an image (.jpg .gif .png): <br />
    <input type="file" name="pic"
           accept="image/gif, image/jpeg, image/png" />
    <br />
    <input type="submit" value="Upload" />
</form>
```

POST request to `upload.cgi` (server-side processing script)

Browser

tutorial3/upload_form.html

HTML Form

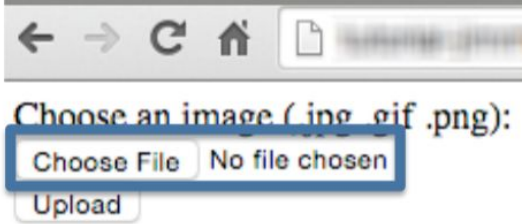
upload_form.html

```
<form enctype="multipart/form-data"
      action="upload.cgi" method="POST">
    Choose an image (.jpg .gif .png): <br />
    <input type="file" name="pic"
           accept="image/gif, image/jpeg, image/png" />
    <br />
    <input type="submit" value="Upload" />
</form>
```

Input type: `file`

Only accept extension `.gif`, `.jpg` and `.png` *(client side checking)*

Browser



tutorial3/upload_form.html

Server-side Script

- How server handle received file ?
  - Apache save it to somewhere
  - Script to write file content to desired location
- Get filename from form parameter

```
filename = form['pic'].filename
```

- Read the file as normal file

```
buf = form['pic'].file.read( )
```

Omit argument → Read whole file
(What if file is very large … ?)

tutorial3/upload.cgi

Server-side Script

- Open a file to write

```
f = open(savePath, 'wb')
```

- Write buffer (read from input) to output file

```
f.write(buf)
```

- Close the output file

```
f.close()
```

tutorial3/upload.cgi

Summary

- **Start earlier !**

- Album display
  - Hardcode images to show
- Image Upload
  - Upload the image
  - Save to appropriate location
  - Read it from browser
  - Generate URL to image

Next week

- ImageMagick
  - Validation
  - Editing
- Database
- Debugging

Thank You