CSCI4140
Open-Source Software Project Development

Tutorial 2

Assignment 1
HTML and Python
Image Upload

22 Jan: Some amendment
- Page 13 example: Should be AttributeError

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

Asg 1

- Languages
  - HTML
  - Python

NO JavaScript is allowed

JavaScript will be blocked during demo

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

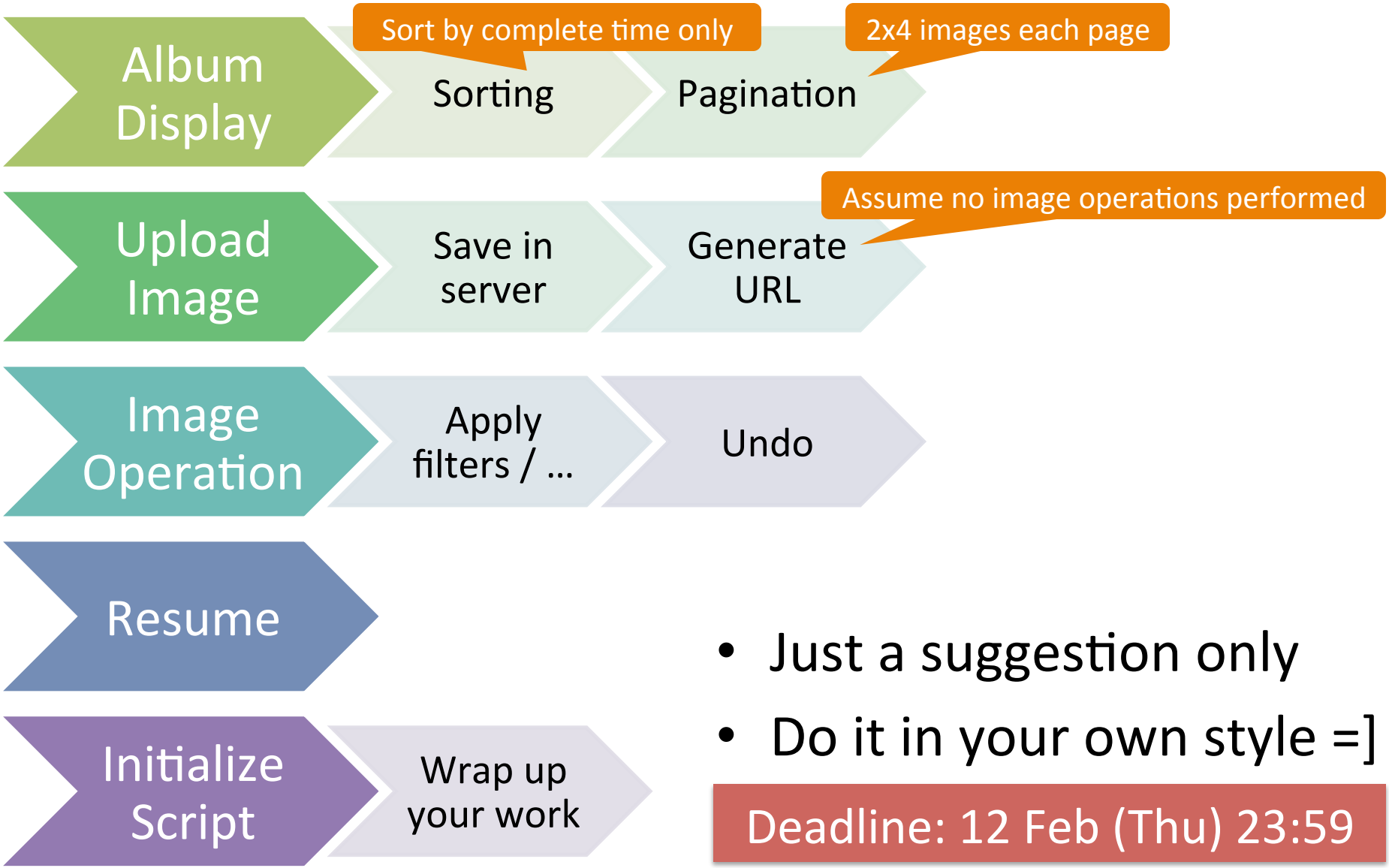
Testing and Demo

- 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

Asg 1



- Just a suggestion only
- Do it in your own style =]

Deadline: 12 Feb (Thu) 23:59

Web Programming

HTML and Python

HTML

- Browser retrieve HTML files (and related files) from web server

- Browser then render the HTML code to a page
  - Show text with style
  - Show images
  - Enable user input

Elements

- HTML file contain HTML elements

```
<tag>content</tag>
```

Start Tag *End Tag*

  - Content can be text, HTML elements or mixture
  - Note the order

```
<tag1>Hello <tag2>World</tag2>!</tag1>
```

- Element with empty content

```
<br />
```

Attributes

- Additional information to an element

- Name-value pairs: `name="value"`

```
<a href="index.html">Index Page</a>
```

Basics

- Some common HTML tags

HTML Tags	Usage
`<html>`	Define root of the HTML document
`<body>`	Define start of page body
` `	Line break
`<table>`	Define a table (use with `<tr>` and `<td>`)
`<div>`	A division / section of page
`<img>`	Show an image (`src` attribute to define location)

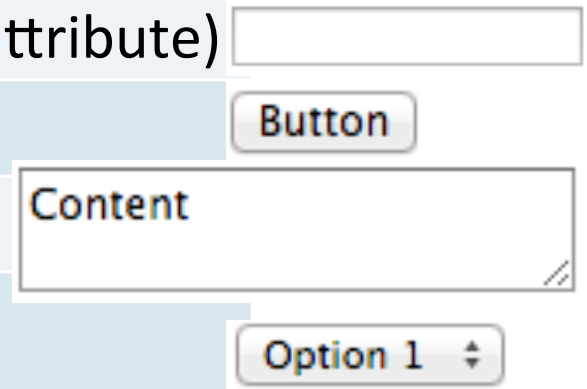
- These tags just define page structure
  - Use `class` / `id` attribute and CSS to define style
- How a page accept user input and send to server ?

More: http://www.w3schools.com/tags/

Forms

- Send data (user input) to server
  - Server receive data and process

`<form>`	Declare HTML a form
`<input>`	Checkbox, text box … (type attribute)
`<button>`	Button
`<textarea>`	Text input field
`<select>` `<option> ...`	Drop-down list



- Use name attribute to distinguish and retrieve in script

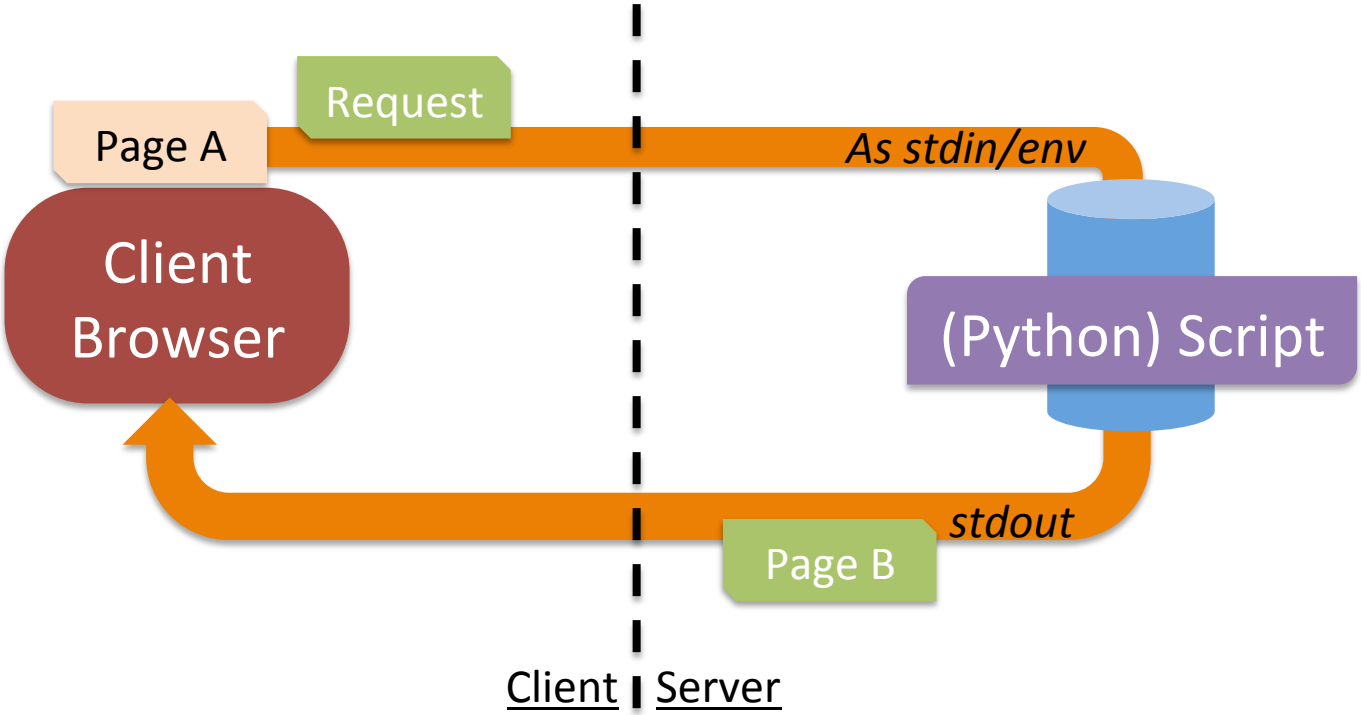
More: http://www.w3schools.com/html/html_forms.asp

tutorial2/html_form.html

Dynamically Generating Webpage

CGI

- 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 execute
    - 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

```
>>> a = 'gg'
>>> a.isupper()
False
>>> a = 4
>>> a.isupper()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'int' object has no attribute 'isupper'
```

Note: We are using Python 2.x

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 …

Tab

Spaces

```
if True:
	print 'tywong'
    print 'sosad'
```

```
>>> 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 semi-colon
  - Semi-colon 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 (1)

- 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 (2)

- Try-except-finally
  - 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

Using Python for CGI

- Regard as printing response to `stdout`

Shebang

HTTP Header

Content (HTML)

helloworld.cgi

```
#!/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

tutorial2/helloworld.cgi

Retrieving Parameters (1)

- `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

`tutorial2/form_process.cgi`

Retrieving Parameters (2)

- `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?*

tutorial2/form_process.cgi

Image Upload

File Upload via HTTP

File Upload

Upload

- File upload using HTTP POST request

- We need (at least) two page 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)

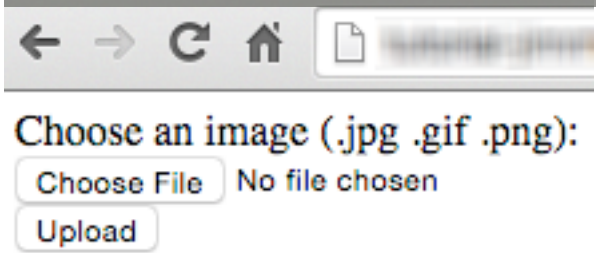
HTML Form (1)

Upload

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>
```

Browser



tutorial2/upload_form.html

HTML Form (2)

Upload

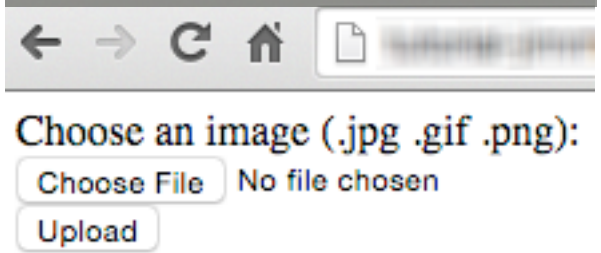
Encoding type: `multipart/form-data` for binary data (file)

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



tutorial2/upload_form.html

HTML Form (3)

Upload

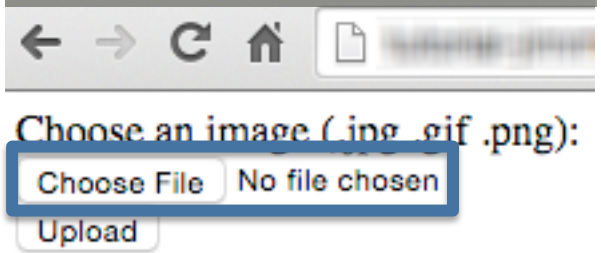
upload_form.html

```
<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



tutorial2/upload_form.html

Server-side Script (1)

Upload

- 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 … ?)

tutorial2/upload.cgi

Server-side Script (2)

Upload

- How server handle received file ?
  - Apache save it to somewhere
  - Script to write file content to desired location
- 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()
```

tutorial2/upload.cgi

Using Persistent Directory

Upload

- Use **persistent directory** to save uploaded files
  - Local changes (include uploaded file) will be flushed when you push your code
- Get the path to persistent storage from environment variable

OPENSHIFT_REPO_DIR	Repository directory path
OPENSHIFT_DATA_DIR	**Persistent directory path**
OPENSHIFT_TMP_DIR	Temporary directory path

- Get the path as following:

```
saveDir = os.getenv('OPENSHIFT_DATA_DIR')
```

More: https://www.openshift.com/page/openshift-environment-variables

Access Persistent Directory

Upload

- Persistent directory cannot access directly from browser

- Create symbolic link from public directory
  - Public directory: `${OPENSHIFT_REPO_DIR}`

- When to create symbolic link ?
  - By running initialize script ?
    - Still a local change
    - Link will be flushed when pushing

More: https://www.openshift.com/developers/deploying-and-building-applications

Action Hooks (1)

Upload

- Action Hooks
  - Scripts run before / after deployment (or other events)
  - Kind of initialize script
    - Still, changes will be flush in next deployment
    - But the script will (re)apply the change just after deployment
- Detail procedure (1) – Add script

```
✔ 02:16:40 jimmy@JimmyHMac ~/openshift/asg1demo master cd .openshi
ft/action_hooks
✔ 02:16:47 jimmy@JimmyHMac ~/openshift/asg1demo/.openshift/action_hooks
master cat > deploy
echo ">> Running deploy script <<"
ln -s ${OPENSHIFT_DATA_DIR} ${OPENSHIFT_REPO_DIR}/data
echo ">> Deploy script done <<"
```

More: http://www.openshift.org/documentation/oo_user_guide.html#action-hooks

Action Hooks (2)

Upload

- Action Hooks
  - Scripts run before / after deployment (or other events)
  - Kind of initialize script
    - Still, changes will be flush in next deployment
    - But the script will (re)apply the change just after deployment
- Detail procedure (2) – Add script to repository

```
✘ 02:17:42 jimmy@JimmyHMac ~/openshift/asg1demo/.openshift/action_hooks
 master chmod +x deploy
✔ 02:17:46 jimmy@JimmyHMac ~/openshift/asg1demo/.openshift/action_hooks
 master git add deploy
✔ 02:17:51 jimmy@JimmyHMac ~/openshift/asg1demo/.openshift/action_hooks
 master ✚ git commit -am "Add deploy script"
[master 03fb6c8] Add deploy script
 1 file changed, 3 insertions(+)
✔ 02:18:06 jimmy@JimmyHMac ~/openshift/asg1demo/.openshift/action_hooks
✔ 02:18:09 jimmy@JimmyHMac ~/openshift/asg1demo/.openshift/action_hooks
 master git push
```

More: http://www.openshift.org/documentation/oo_user_guide.html#action-hooks

Image Upload

- In this tutorial …
  - HTML Form to accept file
  - Write accepted file to persistent directory in server
  - Access your uploaded file on server *from browser*

- Debug tips
  - Check if file saved by your script by SSH to OpenShift server
  - Access persistent directory path by

```
cd ${OPENSHIFT_DATA_DIR}
```

More on OpenShift

SSH to OpenShift application

Environment variables

SSH Access to OpenShift (1)

- OpenShift server can be access via ssh directly
  - Check file upload and save operations
  - Minor change to code for simple bugfix or testing
    - Don't keep coding and testing on server directly
    - Your change will be flush in next deployment
    - Copy your modification to local repository before next push!
  - Access log for error (bug) tracing
    - Detail of log in next tutorial

SSH Access to OpenShift (2)

- Login information can be found in `git clone` URL

```
ssh://54b55a0be0b8cd1aab0000ef@
demo-jimmysinn.rhcloud.com/
~/git/demo.git/
```

Server domain

Login username

```
✔ 01:57:33 jimmy@JimmyHMac ~
ssh 54b55a0be0b8cd1aab0000ef@demo-jimm
ysinn.rhcloud.com
```

```
The authenticity of host 'demo-jimmysinn.rhcloud.com (174.129.86.104)' can't be established.
RSA key fingerprint is cf:ee:77:cb:0e:fc:02:d7:72:7e:ae:80:c0:90:88:a7.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'demo-jimmysinn.rhcloud.com,174.129.86.104' (RSA) to the list of known hosts.
                                                    :
[demo-jimmysinn.rhcloud.com 54b55a0be0b8cd1aab0000ef]\>
```

SSH Access to OpenShift (3)

- Directory structure (extract)
  - Logs can be found here
  - Current deployed version
    - Changes will reflect immediately here
  - Persistent directory

```
app-deployments/
app-root/
    logs/
    repo/
    data/
    ...
git/
perl/
mysql/
phpmyamdin/
```

- Some useful commands
  - `gear [status|stop|start|restart]`
    - Application control
  - `help`

Environment Variables

- OpenShift set some environment variables for you
  - To ease effort in deploying in different applications
  - Avoid conflict when they change the hardware / …

`OPENSHIFT_DATA_DIR`	Persistent directory path
`OPENSHIFT_APP_DNS`	Domain name of application
`OPENSHIFT_APP_NAME`	Name of current application
`OPENSHIFT_MYSQL_*`	MySQL database connection setting *(more in next tuorial)*

  - You can use environment variables in script and shell
  - If some variables missing, try `gear stop` and `gear start`

tutorial2/env-all.cgi

Retrieving Environment Variables

- Two ways to retrieve environment variables
  - As dictionary

What if key not exists?

```
val = os.environ[key]
```

  - By os.getenv

```
os.getenv(key, val)
```

Return val if key not exists
If not specify, val = None

- Remember to `import os`

- Note: you can access query string of GET method / cookies

tutorial2/env.cgi

Summary

- Start doing the assignment 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

Recommendation

- W3School: http://www.w3schools.com/html
  - Provided tutorial on HTML (and more …)
    - Just try their examples
  - Clear reference
  - See also: W3Fool

- Mozilla Developer Network: https://developer.mozilla.org//docs/Web/HTML

END

Contact: Jimmy, Sinn Lok Tsun (Office: SHB115 / SHB1026)

Facebook Group: http://goo.gl/JknhKr

→ Appendix: More on Python

APPENDIX

More on Python

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
```

Modules (2)

- 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

Easter Egg

- Miss the braces?

```
from __future__ import braces
```