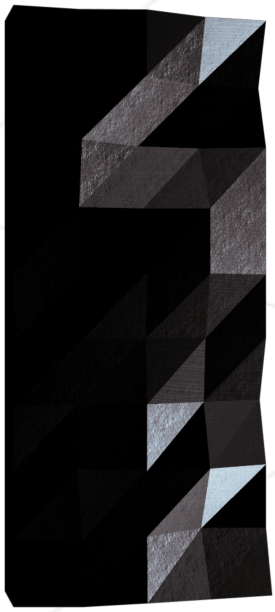

D02 - Django-Python Training

Python Basics 2

Vincent Montécot vmonteco@student.42.fr
42 Sta ff pedago@staff.42.fr

Summary: Today you are going to conquer Silicon Valley through your new skills in OOP with Python!

Contents

Chapter I

Preamble

These are the words of the "Free Software Song": Join us now and share the software; You'll be free, hackers, you'll be free. Join us now and share the software; You'll be free, hackers, you'll be free. Hoarders can get piles of money, That is true, hackers, That Is true. Purpose They Can not help Their neighbors; That's not good, hackers, that's not good. When we-have enough free software At our call, hackers, at our call, We'll kick out Those dirty licenses Ever more, hackers, ever more. Join us now and share the software; You'll be free, hackers, you'll be free. Join us now and share the software; You'll be free, hackers, you'll be free.

Chapter II

Instructions

- Only this page serve as a reference: Do not fi rm the hallway noise.
- The subject can change up to an hour before rendering.
- If no contrary information is explicitly present, you should assume that the versions of languages and frameworks used are (or later):
  - Python 3
  - HTML5
  - CSS 3
  - Javascript EM6
  - Django 1.9
  - psycopg2 2.6
- Unless otherwise indicated in the subject, the fi les in python each year on
  Python one (d01, d02 and d03) must have their end block if __name__ == '__main__': in order to insert the entry point in the case of a program, or tests in the case of a module.
- Unless otherwise indicated in the subject, each year days on
  Django will have its own application in the project to make for reasons peda- gogiques.
- The exercises are precisely ordered from simple to more complex. In no event shall we bear no attention or take into account a complex exercise if a simpler exercise is not very successful.
- Attention to the rights of your fi les and your directories.
- You must follow the rendering process for all your exercise: only the present work on your GIT repository will be evaluated in defense.
- Your exercises will be evaluated by your pool mates.
- You should leave in your repertoire no other fi le than those explicitly speci fi ed by the forward drills.
- Unless otherwise specified in the topic you should not include in your rendering:

- The files __ pycache__.
- Any migration.
  Warning, you are still advised to make the fi le migration / __ init__.py,
  it is not necessary, but simplifies the construction of migration. Do not add this fi le will not invalidate your rendering but you *must* be able to manage migration for correction in this case.
- The file created by the command collectstatic of manage.py (with the way the value of the variable STATIC_ROOT).
- The fi le Python bytecode (fi les with extension. pyc).
- database fi les (especially with sqlite).
- Any fi le or folder must or can be created by the normal behavior of the rendering work.

It is recommended that you modify your. gitignore in order to avoid accidents.

- When you need to get an accurate output in your programs, it is of course forbidden to a ffi expensive output precalculated instead of performing the exercise cor- rectly.
- You have a question ? Ask your neighbor to the right. Otherwise, try your left neighbor.
- Your reference manual called Google / man / Internet / ....
- Think discuss on the forum of your pool Intra!
- Read the examples carefully. They might require things that are not otherwise specified in the subject ...
- Please, by Thor and Odin! Re fl échissez dammit!

chapter III

Specific Rules of the day

- No code in the global scope. Make functions!
- Unless otherwise indicated, all written in Python fi les will end with a block

```
if __name__ == '__main__':
        #   Your tests and your error handling
```

- Any exception catchée not invalidate the work, even in a case of error you are asked to test.
- No authorized import, except for those explicitly mentioned in the 'Authorized Functions' of the cartridge each year ..

Chapter IV

Exercise 00

	Exercise: 00
Exercise 00: Conquering the Silicon Valley!	
Render Folder: *ex* 00 / Files to make: render.py, myCV.template, settings.py	
Permitted functions: import sys, os, re	
Remarks: n / A	

You have fi nished your super developer training and a new life full of S'o ff ers prospects to you. Arriving in Silicon Valley, you have an idea in mind: develop your idea of revolutionary CV generator using a pioneering technology, and become the new Bill Gates of the job search.

It only remains to develop the technology. Make a program render.py which will take a file with an extension. template

parameter. This program will read the contents of the file, change some patterns with defined values in a file settings.py (The presence of a block if __name__ == '__main__': is not required for this fi le) and write the result to a file with the extension. html.

The following example *will* be exactly reproduced with your program.

```
$> Cat settings.py: name =
"duoquadragintian" $> cat file.template <p>
"- Who are you?

- A {name}! "</ P>
$> $ Python3 render.py file.template> cat file.html <p> "-
Who are you?

- A duoquadragintian! "</ P>
```

The mistakes, including poor fi le extension, a file does not exist or a bad number of arguments to be managed.

You must make a file myCV.template which, when converted to HTML fi le, should at least contain the complete structure of a page (doctype, head and body), the page title, the name and surname of the owner of CV, age, and profession. Of course, this information should not appear directly in the fi le

. template.

Chapter V

Exercise 01

	Exercise: 01
01 Exercise: Innovative Startup seeks intern. 10 years of experience required. Render Folder: *ex* 01 / files	
to make: intern.py	
Authorized Functions: Notes: n	
/ A	

You can not go it alone on such an adventure. You decide to recruit someone to do ter coffee assist you, preferably a trainee (it's ~~cheaper~~).

Realize the class Intern containing the following features:

A builder taking a string parameter and assigning its
an attribute value Name. A default " My name? I'm nobody, an intern, I have no name. " will be implemented.

A method __ str __ () which will return the attribute name of the proceedings.

A class Coffee with a simple method __ str __ () which will return the string
character " This is the worst coffee you ever tasted. ".

A method work () which will lift just one exception (use type (Exception)
Basic) with the text " I'm just an intern, I can not do that ... ".

A method make_coffee () which will return an instance of the class Coffee than
you implemented in the classroom Intern.

In your tests, you need to instantiate the class twice Intern once nameless, and once with the name "Mark".

A ffi in the name of each instance. Ask Mark make you a coffee and AF-fi in the result. Ask the other student to work. You **must** handle the exception in your test.

Chapter VI

Exercise 02

	Exercise 02 Exercise 02: 5
classes 1 cup.	
Render Folder: *ex* 02 / Files to make: beverages.py	
Authorized Functions: Notes: n	
/ A	

The coffee is good but in the long run it's a bit boring to not have more choices. Make a class HotBeverage with the following features:

An attribute price a value of 0.30.

An attribute name with the value "Hot beverage".

A method description() returning a description of the proceedings. The value
the description will be " Just some hot water in a cup. ".

A method __ str __ () returning a description of the proceedings in this form:

```
name: <name attribute>
price: <price attribute limited to two decimal point> Description: <instance's description>
```

eg a ffi chage an instance of HotBeverage give:

```
name: hot beverage price: 0.30

Description: Just some hot water in a cup.
```

Realize then derived classes HotBeverage following:

Coffee:

name: " coffee "

price: 0.40

description: " A coffee, to stay awake. "

Tea:

name: " tea "

price: 0.30

description: " Just some hot water in a cup "Chocolate.:

name: " chocolate "

price: 0.50

description: " Chocolate, sweet chocolate ... "Cappuccino:

name: " cappuccino"

price: 0.45

description: "Un po 'di Italia nella sua Tazza"

You must redefine THAT what is necessary. (Cf. DRY)

Instantiate in your tests from each class: HotBeverage, Coffee, Tea, Chocolate and Cappuccino and a ffi in.

Chapter VII

Exercise 03

	Exercise: 03
Exercise 03: Glorious co ff ee machine!	
Render Folder: *ex* 03 / Files to make: machine.py, beverages.py	
Permitted functions: import random	
Remarks: n / A	

That's it, your company is on! You now have a local acquired through your first fundraiser, a trainee to make coffee and a level of green plant 10 to the entrance of the building to keep it all.

However it must be admitted: coffee produced by your intern is rubbish and a half minimum wage per month is a bit expensive for the sock juice. This is the time of in- vestir in new equipment necessary for your professional success!

Realize the class CoffeeMachine containing:

- A builder.
- A class EmptyCup inheriting HotBeverage, with the name " empty cup "
  as prices 0.90 and the description " An empty cup ?! Gimme my money back! ".
  Copy the fi le beverages.py last year in the record of this exercise to use the classes it contains.
- A class BrokenMachineException inheriting Exception with the text " This coffee machines Has to be repaired. ". This text must be defined in the constructive tor of the exception.
- A method repair () which repairs the machine for it to be used to nou- calf hot drinks.
- A method serve () which will have the following characteristics:

  Settings : A single parameter (other than self) which will be a derived class
  of HotBeverage.

  return: Once two (randomly), the method returns an instance of

last classroom setting, and half the time an instance of EmptyCup.

obsolescence: The machine is not of the best quality and therefore falls failure after 10 drinks served.

In case of failure : the method call serve () should cause the removal of a exception type CoffeeMachine.BrokenMachineException up to-that the method repair () is called.

repair: After a call to the repair (), the method serve () can again operate without exception thrown during a cycle of 10 drinks before falling down.

In your tests, instantiate the class CoffeeMachine. Ask various drinks ing ve- fi le beverages.py and a ffi in the drink you used the machine until it fails it (you *must* handle the exception then lifted). Fix the machine and start to do the machine breaks down again (except manage again).

Chapter VIII

Exercise 04

	Exercise: 04
Exercise 04: A base class ft. RMS.	
Render Folder: *ex* 04 / Files to make: elem.py	
Authorized Functions: Notes: n	
/ A	

Now is the time to improve your presence on the web. You would like to use your new skills in Python for modeling e ff ectively your HTML content, but you want to receive the advice of a superior being to find out how. You decide to o ff er your intern sacrifice that the gods of programming.

Now that you have a machine to make coffee, you do not really need it ... You therefore immolate.

Saint IGNUcius you will appear to you a revelation:

> *"HTML elements share more or less the same structure (tag, content, attributes). It would be wise to conduct a class capable of Ras seem all these behaviors and characteristics common to then use the power of inheritance in Python to derive easily and sim- plement this class without having to rewrite everything. "*

Then St. IGNUcius see the Mac on which you work. Taking fear, he fled without giving you more details, leaving behind only a fi le testing. Without hesitation, you realize the class Elem with the following characteristics:

- A builder may take as parameter the name of the element, its attributes HTML, its content and the element type (single or double tags).
- A method __ str __ () returning the code HTML of the element.
- A method add_content () to add items to the end of the content. If you are doing your job well, you will be able to represent any element HTML and content with your class Elem. Home stretch :

- the fi le tests.py tarball provided in the annex of the subject **must** operate cor- rectly (no assertion error, the output of the test explicitly announcing his success). Obviously, we are not cruel enough to test the functionalities that are not explicitly requested in this exercise. Hahaha ... No, we are not, I assure you.

- You will also need to reproduce and expensive ffi the structure with your class Elem:

```
<Html>
    <Head>
        <Title>
            "Hello ground!" </ Title>
     </ head> <body>


        <H1>
            "Oh no, not again!" </ H1>

        <Img src = "http://i.imgur.com/pfp3T.jpg" /> </ body> </ html>
```

Chapter IX

Exercise 05

	Exercise: 05
Exercise 05: Make your own items!	
Render Folder: *ex* 05 / Files to make: elem.py, elements.py	
Authorized Functions: Notes: n	
/ A	

Congratulations! You are now able to generate any HTML element and its contents. However, this is a bit cumbersome to generate each item by stating where each attribute to each instantiation. This is an opportunity to **use inheritance to other little easier to use classes. Perform the following classes derived from your class Elem last** year:

- html, head, body
- title
- meta
- img
- table, th, tr, td
- ul, ol, li
- h1
- h2
- p
- div
- span
- hr
- br

The constructor of each class will be able to take content first argu- ment, as well:

```
print (Html ([Head () Body ()]))
```

a ffi chera well:

```
<Html>
    <Head> </ head>
  <body> </ body> </
  html>
```

Be smart and reuse functionality you encoded in the class Elem. You **must** use inheritance.

Demonstrate the operation of these classes for testing your desired number su ffi cient to cover all the features. After having addressed these classes, you will not need to specify the name or type of a tag, which is very convenient. Therefore you should never instantiate your class Elem, It is now **not allowed.**

To help you understand the advantage derived classes Elem compared to the direct use Elem, resume the document structure HTML the previous exer- cise. You must reproduce it using your new classes.

```
<Html>
    <Head>
        <Title>
            "Hello ground!" </ Title>
      </ head> <body>


        <H1>
            "Oh no, not again!" </ H1>

        <Img src = "http://i.imgur.com/pfp3T.jpg" /> </ body> </ html>
```

It's much more simple, right? :)

Chapter X

Exercise 06

	Exercise 06 Exercise 06:
Validation	
Render Folder: *ex* 06 / Files to make: Page.py, elem.py, elements.py	
Authorized Functions: Notes: n	
/ A	

Despite real progress in your work, you would like that everything is a little cleaner. A little framed, you are like that: you like the constraints and challenges. So why not impose a standard structure for your documents HTML? Commencez by copying the classes of the two previous years in the record of this exercise.

Create a class Page which the manufacturer must take as a parameter an instance of a class inheriting Elem. Your class Page must implement a method isValid ()
which must return true if all the following rules repectées and false if not :

- If during the course of the tree, a node is not type html, head, body, title, meta, img, table, th, tr, td, ul, ol, li, h1, h2, p, div, span, hr br or
  text, the shaft is invalid.
- html must contain exactly Head, then a Body.
- Head should only contain a single title and only this Title.
- Body and div only contain elements of following type: H1, H2, Div, table, ul, ol, Span, or Text.
- Title, H1, H2, Li, Th, Td must not contain a single text and only this
  Text.
- P should contain only Text.
- span should contain only text or some P.
- IU and ol] must contain at least one Li and only Li.

- **Tr must contain at least one Th or td and only Th or some Td. The Th and the td must be mutually** exclusive.
- **table: should contain only Tr and only Tr.**

Your class Page must also be able to:

- **A ffi expensive its code HTML when it print an instance. Warning: the code HTML**
  a ffi ket must be preceded by a doctype if and only if the type of the root element Html.

- **Write the code HTML in a file with a method write_to_file which takes as a parameter the name of the file.**
  Warning: the code HTML written in the fi le must be preceded by a doctype if and only if the type of the root element
  Html.

Demonstrate how your class Page by tests of your choice by many su ffi cient to cover all the features.