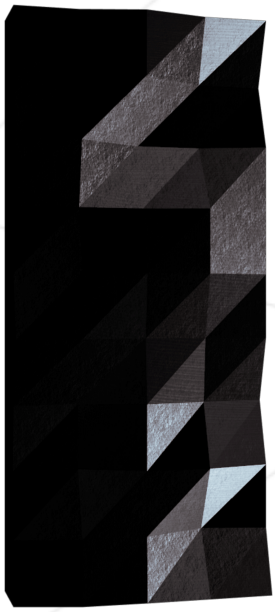

D02 - Python-Django training

Python 2 basics

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

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

Contents

I Preamble 2

II Instructions 3

III Specific rules of the day 5

IV Exercise 00 6

V Exercise 01 8

VI Exercise 02 9

VII Exercise 03 11

VIII Exercise 04 13

IX Exercise 05 15

X Exercise 06 17

Chapter I

Preamble

Here are the lyrics to 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. But they cannot 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 will serve as a reference: do not be fooled by noise in the hallway.
- The subject can change up to an hour before rendering.
- If no contrary information is explicitly present, you must assume that the versions of the languages and frameworks used are the following (or later):
  - Python 3
  - HTML5
  - CSS 3
  - Javascript EM6
  - Django 1.9
  - psycopg2 2.6
- Unless otherwise indicated in the subject, the python fi les for each exercise on **Python only (d01, d02 and d03) must include at their end a 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 exercise of the days relating to **Django will have its own application in the project to be returned for educational reasons.**
- The exercises are very precisely ordered from the simplest to the most complex. In no case will we pay attention to or take into account a complex exercise if a simpler exercise is not perfectly successful.
- Pay attention to the rights of your fi les and directories.
- You must follow the rendering procedure for all your exercises: only the work present on your GIT repository will be evaluated in defense.
- Your exercises will be evaluated by your pool mates.
- You must not leave any other fi les in your repertoire other than those explicitly speci fi ed by the exercise statements.
- Unless otherwise specified in the subject you must not include in your rendering:

- The files __ pycache__.
- Possible migrations.
  Please note, you are still advised to return the fi le migrations / __ init__.py,
  it is not necessary but simplifies the construction of migrations. Not adding this fi le will not invalidate your rendering but you *must* be able to manage your migrations in correction in this case.
- The folder created by the command collectstatic of manage.py (with the value of the variable as path STATIC_ROOT).
- Files in Python bytecode (Files with an extension in. pyc).
- The database fi les (especially with sqlite).
- Any fi le or folder before or which may be created by the normal behavior of the rendered job.

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

- When you need to obtain a precise output in your programs, it is of course forbidden to display a pre-calculated output instead of performing the exercise correctly.
- You have a question ? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual is called Google / man / Internet / ....
- Consider discussing on the Swimming pool forum of your Intra!
- Read the examples carefully. They may well require things that are not otherwise specified in the subject ...
- By pity, by Thor and by Odin! Think name of a pipe!

Chapter III

Specific rules of the day

- No code in the global scope. Do functions!
- Unless otherwise specified, all your fi les written in Python must end with a block.

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

- Any exception not caught will invalidate the job, even in an error case that you are asked to test.
- No import allowed, except for those explicitly mentioned in the 'Authorized Functions' section of the title block of each exercise.

Chapter IV

Exercise 00

	Exercise: 00
Exercise 00: Conquering Silicon Valley!	
Render folder: *ex* 00 / Files to be returned: render.py, myCV.template, settings.py	
Authorized functions: import sys, os, re	
Notes: n / A	

You have completed your awesome developer training and a new life full of prospects awaits you. When you arrive in Silicon Valley, you have just one thing in mind: to develop your idea of a revolutionary resume generator using cutting-edge technology, and become the new Bill Gates of job search.

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

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

The following example *will have to* be able to 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>
```

Errors, such as a bad fi le extension, a fi le that does not exist, or the wrong number of arguments, will need to be handled.

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

. template.

help (globals), keyword expansion ...

Chapter V

Exercise 01

	Exercise: 01
Exercise 01: Innovative start-up looking for an intern. 10 years of experience required. Render folder: *ex* 01	
/ Files to be returned: intern.py	
Authorized Functions: Notes: n	
/ A	

You cannot embark on such an adventure alone. You decide to hire someone to make the coffee to assist you, preferably an intern (it's ~~cheaper~~).

Complete the class Intern containing the following features:

A builder taking a character string as a parameter and assigning its value to an attribute Name. A default value " 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 instance.

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

A method work () which will just throw an exception (use the (Exception) type base) with the text " I'm just an intern, I can't do that ... ".

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

In your tests, you must instantiate the class twice Intern, once unnamed, and again with the name "Mark".

A ffi in the name of each instance. Ask Mark to make you a coffee and display the result. Ask the other intern 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 be returned: beverages.py	
Authorized Functions: Notes: n	
/ A	

Coffee is good but in the long run it's a bit boring not to have more choice. Complete a class HotBeverage with the following features:

An attribute price a value of 0.30.

An attribute name with value “Hot beverage”.

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

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

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

for example displaying an instance of HotBeverage would give:

```
name: hot beverage price: 0.30

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

Then carry out the classes derived from 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 should redefine ONLY what is necessary, what you need to change, to redefine ... (cf. DRY).

Instantiate in your tests each of the classes among: HotBeverage, Coffee, Tea, Chocolate and Cappuccino and has ffi at the.

Chapter VII

Exercise 03

	Exercise: 03
Exercise 03: Glorious coee machine!	
Render folder: *ex* 03 / Files to be returned: machine.py,	
beverages.py	
Authorized functions: import random	
Notes: n / A	

That's it, your company is launched! You now have premises acquired thanks to your first fundraiser, an intern to make the coffee and a level 10 green plant at the entrance of the building to keep everything.

However, it must be admitted: the coffee produced by your trainee is foul and half the minimum wage per month is a bit expensive for sock juice. Now is the time to invest in the new equipment necessary for your professional success!

Complete the class CoffeeMachine containing:

- A builder.
- A class EmptyCup inheriting from HotBeverage, with for name " empty cup ",
  as price 0.90 and as description " An empty cup ?! Gimme my money back! ".
  Copy the fi le beverages.py from the previous exercise in the file for that exercise to be able to use the classes it contains.
- A class BrokenMachineException inheriting from Exception with for text " This coffee machine has to be repaired. ". This text must be defined in the constructor of the exception.
- A method repair () who repairs the machine to enable it to serve hot drinks again.
- 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 in two (randomly), the method returns an instance of

the class passed as a parameter, and once in two an instance of EmptyCup.

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

In case of failure : calling the method serve () must cause the lifting of a type exception CoffeeMachine.BrokenMachineException until the method repair () be called.

Repair: After a method call repair (), the method serve () can again operate without raising the exception for a cycle of 10 drinks, before failing again.

In your tests, instantiate the class CoffeeMachine. Ask for various drinks from the fi le beverages.py and keep the drink that the machine serves you until it breaks down (you *must* manage the exception then raised). Repair the machine and start over until the machine fails again (handle the exception again).

Chapter VIII

Exercise 04

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

Now is the time to improve your presence on the WEB. You would love to use your newfound knowledge of Python to effectively model your HTML content, but you would like to receive advice from a higher being on how to do it. You decide to offer your intern in sacrifice to the gods of programming.

Now that you have a machine to make the coffee, you don't really need it anymore ... So you immolate it.

Saint IGNUcius then appears to you to make you a revelation:

> *"HTML elements share more or less the same structure (tag, content, attributes). It would be a good idea to make a class that can bring together all of these common behaviors and characteristics and then use the power of inheritance in Python to easily and simply derive that class without having to rewrite everything. "*

It is then that St. IGNUcius sees the Mac you are working on. Getting scared, he runs away without giving you more details, leaving behind only a test fi le, as well as an incomplete class. Without hesitation, you complete the class Elem (the holes to be filled being indicated by [...]) which will have the following characteristics:

- A constructor that can take as parameter the name of the element, its attributes HTML, its content and the type of element (single or double tags).
- A method __ str __ () returning the code HTML of the element.
- A method add_content () allowing you to add elements at the end of the content.
- A subclass of *Exception* within it.

If you do your job well, you will be able to represent any element HTML and its content with your class Elem. Home stretch :

- the fi le tests.py provided in the tarball attached to the subject **must** function correctly (no assertion error, the exit of the test explicitly announcing its success). Obviously, we are not cruel enough to test features that are not explicitly called for in this exercise. Hahaha ... No, we are not, I assure you.

- You will also need to reproduce and display the following structure using 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 elements!	
Render folder: *ex* 05 / Files to be returned: elem.py,	
elements.py	
Authorized Functions: Notes: n	
/ A	

Congratulations! You are now able to generate any HTML element and its content. However, it is a bit cumbersome to generate each element, specifying each attribute each time at each instantiation. This is an opportunity to use inheritance to make other small classes easier to use. Make the following classes derived from **your class Elem from the previous 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 should be able to take the content as the first argument, thus:

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

will look for:

```
<html>
    <head> </head>
  <body> </body> </html>
```

Be smart and reuse the features you coded in the Elem class. You **must** use inheritance.

Demonstrate how these classes work with tests of your choice in sufficient number to cover all functionalities. After coding these classes, you will no longer need to specify the name or type of a tag, which is very convenient. You should therefore never directly instantiate your class again. Elem, it's now **not allowed.**

To help you fully understand the advantage of derived classes from Elem compared to direct use from Elem, let's take the structure of the document HTML of the previous financial year. You have to 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 simpler, isn't it? :)

Chapter X

Exercise 06

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

Despite real progress in your work, you would like everything to be a little cleaner. A little more framed, you are like that: you like constraints and challenges. So why not impose a standard on the structure of your documents HTML? Start by copying the classes from the previous two exercises into this exercise's folder.

Create a class Page whose constructor must take as parameter an instance of a class inheriting from Elem. Your class Page must implement a method isvalid ()
who should return True if all the following rules are respected, and False if not :

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

Your class Page must also be able to:

- View your code HTML when we print an instance. Attention: the code HTML displayed must be preceded by a doctype if and only if the type of the root element is Html.

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

Demonstrate how your classroom works Page by tests of your choice in sufficient number to cover all functionalities.