Piscine 101

Python 02

Summary: This document is the subject for the PYTHON module 02 of the Piscine 101 @ 42Tokyo.

Contents

Chapter I

Instructions

- Only this page will serve as reference; do not trust rumors.
- Watch out! This document could potentially change up to an hour before submission.
- These exercises are carefully laid out by order of difficulty - from easiest to hardest. We `will not` take into account a successfully completed harder exercise if an easier one is not perfectly functional.
- Make sure you have the appropriate permissions on your files and directories.
- You have to follow the submission procedures for every exercise.
- Your exercises will be checked and graded by your fellow classmates.
- On top of that, your exercises will be checked and graded by a program called Moulinette.
- Moulinette is very meticulous and strict in its evaluation of your work. It is entirely automated and there is no way to negotiate with it. So if you want to avoid bad surprises, be as thorough as possible.
- Exercises in Shell must be executable with /bin/sh.
- You cannot leave any additional file in your directory than those specified in the subject.
- Got a question? Ask your peer on the right. Otherwise, try your peer on the left.
- Your reference guide is called `Google / man / the Internet / ....`
- Examine the examples thoroughly. They could very well call for details that are not explicitly mentioned in the subject...
- If no other explicit information is displayed, you must assume the following versions of languages : `Python - python3.8.2`.
- No code in the global scope(only import). We want functions!
- you're free to define as many function as you like and name them as you like also.

- Each turned-in file must end with a function call in a condition identical to:

```
if __name__ == '__main__':
    your_function( whatever, parameter, is, required )
```

- You can set an error management in this condition.
- No import will be authorized, except the ones explicitly mentioned in the 'Autorized functions' in each exercise's description.
- You won't have to manage the exceptions raised by the `open` function.

Chapter II

Foreword

```
Brendan : Hi, you there! Mysterious stranger! Yoohoo!
Brendan : Year, you! There any other mysterious stranger around here?
Brendan : You know what? How 'bout I tell you a joke?
      V : Sure, be my guest.
```

Chapter III

Exercise 00 : Buy this product!

	Exercise 00
a very capable salesperson sells anything he wants	
Turn-in directory : *ex*00/	
Files to turn in : `salesperson.py, main.py`	
Allowed functions : `n/a`	

You cannot start this journey alone. You choose to recruit someone to sell your products, a sales person would be better.

Create the `Salesperson` class containing the following functionalities:

- `A builder` taking a string as a parameter, assigning its value to a name attribute. "nameless salesperson." will be implemented as default value.
- `A method __str__()` that will return name attribute of the instance.
- `A Product` class with a simple ___str___() method that will return the string "This is the best product in the world.".
- `A promote()` method that will return an instance of the Product class that you will have implemented in Salesperson class.

Turn in your `main.py` file to test this exercise. Your main.py `will be reviewed` at the time of your defense.

Chapter IV

Exercise 01 : Be more specific!

	Exercise 01
Because no one buys it	
Turn-in directory : *ex*01/	
Files to turn in : `beverage.py, main.py`	
Allowed functions : `n/a`	

Create the `Product` class containing the following functionalities:

- `A builder` taking a string, a float, and another string as parameters, assigning its value to a Name attribute, a price attribute, and a description attribute. Use a value of your choice to be implemented as default value.
- `A __str__()` method returning an instance description in this form:

  ```
  <name attribute> : <description attribute>
  ```

  for example,

  ```
  the best protein powder : Quite self explanatory, isn't it?
  ```

- `A print_attr()` method that will print all 3 attributes in this form:

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

  for example,

  ```
  name : the best protein powder
  price : 99.99
  description : Quite self explanatory, isn't it?
  ```

Create the inherited `Beverage` class from the Product class, containing the all the Product class functionalities using a function `super()`:

- `A builder` taking a float and all the Product class attribute as parameters, assigning its value to a temperature attribute in addition to name, price, and description attribute. Use a value of your choice to be implemented as default value.

- A `print_attr()` method that will print all 4 attributes in this form:

```
name : <name attribute>
price : <price attribute limited to two decimal points>
description : <description attribute>
temperature : <temperature attribute>
```

Turn in your `main.py` file to test this exercise. Your main.py `will be reviewed` at the time of your defense.

Chapter V

Exercise 03 : Sell them now.

	Exercise 03
Brendan the Vending Machine	
Turn-in directory : *ex*03/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `sys`	

Create the `Vendingmachine` class containing the following functionalities:

- `A builder` taking a string, an array of Beverage instances as parameters, assigning its value to a name attribute and a stock attribute.
- `A __str__()` method returning an instance description in this form:

  ```
  <name attribute> the vending machine
  ```

- `A sell()` method that takes 1 argument called `beverage_name`, and print a message of choice. Differ the message when there is a Beverage instance named same as product_name, and not.
- `A ask()` method that will print a message to ask the beverage name, store the user input, then call `sell(beverage_name)` function.

for example, the `ask()` method can be something like this:

```
Hello, what would you like?
coffee
Here is your coffee!

Hello, what would you like?
milk shake
Sorry! I do not have milk shake...
```

Turn in your `main.py` file to test this exercise. Your main.py `will be reviewed` at the time of your defense.

Chapter VI

Exercise 04 : Sell them more.

	Exercise 04
better UX	
Turn-in directory : *ex*04/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `n/a`	

Create the `Vendingmachine` class containing the following functionalities:

- `A builder` taking 2 string, an array of Beverage instances as parameters, assigning its value to a name, a greeting, and a stock attribute.

  For example,

  ```
  vm = Vendingmachine("Henry", "Hi, I am Henry", [Beverage('coca-cola', 1, 'the most popular drink
      !', 3)])
  ```

- `A greet()` method which print its greeting attribute.
- `A sell()` method that takes 1 argument called `beverage_name`, and print a message of choice. Differ the message when there is a Beverage instance named same as product_name, and not.
- `A display()` method that will show its lineup of stocks.
- `A random_recommend()` method that will choose a random instance from its stock, then print a recommendation.
- `A ask()` method from the previous exercise, yet greet() and display() before asking a name of beverage.
- `A sell()` method from the previous exercise, yet random_recommend() when the given beverage name is not found in its stock.

Example of Henry the vending machine's `ask()` method:

```
Hi. I am Henry.

I have...
```

```
coffee : It is a must for brainwork!
coca-cola : the most popular drink!

would you like something?
matcha

sorry, I don't have the product called matcha
I recommend...
coca-cola : the most popular drink!
```

Turn in your `main.py` file to test this exercise. Your main.py `will be reviewed` at the time of your defense.

Chapter VII

Exercise 05 : Can't be infinite.

	Exercise 05
finite stock	
Turn-in directory : *ex*05/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `n/a`	

Add or rewrite the following functionalities to the previous `Vendingmachine` class:

- `A builder` now takes an array of tuples instead of Beverage instances for the stock attribute. The each tuple is consist of Beverage instances and integer, which indicates how many drinks are in the stock.

  For example, one can instantiate a Vendingmachine as follows:

  ```
  coffee = Beverage('coffee', 100, 'It is a must for brainwork!!', 80)
  coca = Beverage('coca-cola', 100, 'the most popular drink!', 3)
  vm = Vendingmachine("Henry", "Hi, I am Henry", [(coca, 1),(coffee, 5)])
  ```

- `A sell()` method from the previous exercise, yet it sells the beverage if there is any stock left and it reduce the stock when it is sold.

- `A add_stock()` method takes 2 parameters, a string and integer, as name of a beverage and number of stock. This method increase the number of stocks when when the Beverage instance with the same name exist in the stock. This method will create a new Beverage instance and add it into its stock with the given number of stock if the same-name Beverage instance is not present in the current stock.

Turn in your `main.py` file to test this exercise. Your main.py `will be reviewed` at the time of your defense.

Chapter VIII

Exercise 06 : I am your manager.

	Exercise 06
mangers manage their stock	
Turn-in directory : *ex*06/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `n/a`	

Add or rewrite the following functionalities to the previous `Vendingmachine` class:

- `A builder` taking another string as a parameter in addition to all the previous parameters. This parameter should be assigned to a manager_code attribute. For example,

```
vm = Vendingmachine("Henry", "Hi, I am Henry", [(coca, 1),(coffee, 5)], "I am your manager.")
```

- `A ask()` method from the previous exercise, yet inputting a string "manager mode" instead of beverage name will not sell a beverage but start its manager mode, calling a method manager_mode.

- `A manager_mode()` method will prompt the manager_code to the user, and exit whenever the input does't match the manager_code attribute. After the code has matched, prompt a name of beverage and the number of stock to add. Then, call the previously made method add_stock(). If "show stock" is inputted instead of a beverage name, this method will call another method show_stock() instead.

- `A show_stock()` method displays all the stocked beverage with each number of stock in the form of your choice.

Example of Henry the vending machine's manager mode:

```
would you like something?
manager mode

Please tell me the manager code:
I am your manager.

Confirmed...
Do you want to show stock or add a stock? Type "show stock" to show my stock or a beverage name to add
    stock.
```

```
show stock

Here is my current stock:
coffee : It is a must for brainwork! ---> 2 left
coca-cola : the most popular drink! ---> 5 left
```

Turn in your `main.py` file to test this exercise. Your main.py `will be reviewed` at the time of your defense.

Chapter IX

Exercise 07 : Bonus

	Exercise 07
The more original, the better	
Turn-in directory : *ex*07/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `n/a`	

Make your vending machine class more original and better. Here are some ideas for the bonus. you can choose from the below, or use your imagination.

- `A recommend()` method that recommends a beverage yet not randomly and more strategically. For example, it can recommend the beverage with the largest number of stock.
- `A safe` attribute where a vending machine can calculate the earnings from the price attribute of the beverages sold.
- `A say()` method that will print the characters slowly so that a vending machine can be seen more humane. 1 letter per 0.1 seconds for example.
- `A say()` method that will print the characters slowly so that a vending machine can be seen more humane. 1 letter per 0.1 seconds for example.
- `A lottery()` method which tries one's luck, and reward a drink when one wins the lottery.
- Add the visual of a vending machine, a formatted display and a face to show the friendliness of the machine.

Turn in your `main.py` file to test this exercise. Your main.py `will be reviewed` at the time of your defense.