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.
- 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.
- You cannot leave any additional file in your directory than those specified in the subject.
- 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 - python3.9.0`.
- No import will be authorized, except the ones explicitly mentioned in the 'Autorized functions' in each exercise's description and your own modules.
- You are allowed to use any function.
- You should use guacamole.42tokyo.jp to validate exercises.

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	
Turn-in directory : *ex*00/	
Files to turn in : `salesperson.py, product.py, main.py`	
Allowed functions : `n/a`	

You created a most powerful energy drink in the world. It's time to sell this product!

- Create the `A Product` class as follows inside product.py:

```
class Product:
   def __str__(self):
       """
       return a string such as 'This is the best product in the world.'
       """
       pass
```

  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 as follows inside salesperson.py:

```
class Salesperson:
    def __init__(self, name='nameless salesperson.'):
        """
        assign its value to a name attribute.
        """
        pass

    def __str__(self):
        """
        return name attribute of the instance.
        """
        pass

    def promote(self):
        """
        return a Product instance.
        """
        pass
```

- 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, beverage.py, main.py`	
Allowed functions : `n/a`	

Your catch phrase charm every customer! Unfortunately, You haven't decide the price of your product. Let's set up the price.

- Create the `Product` class as follows inside product.py:

```
class Product:
    def __init__(self, name, price, description):
        """
        assign its values to a name attribute, price attribute, and a description attribute.
        """
        pass

    def __str__(self):
        """
        return its name and description attributes in the format.
        <name attribute> : <description attribute>
        """
        pass

    def print_attr(self):
        """
        return its name, price and description attributes in the format.
        name : <name attribute>
        price : <price attribute limited to two decimal points>
        description : <description attribute>
        """
        pass
```

- Create the inherited `Beverage` class from the Product class, containing the all the Product class functionalities using a function `super` inside beverage.py:

```
class Beverage(Product):
    def __init__(self, name, price, description, temperature):
        """
        assign its values to name, price, description, temperature attribute, using super(),
        """
        pass

    def print_attr(self):
        """
        return its name, price, description and temperature attributes as a string in the format
              below.
        name : <name attribute>
        price : <price attribute limited to two decimal points>
        description : <description attribute>
        temperature : <temperature attribute>
        """
        pass
```

- 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 02 : Sell them now.

	Exercise 02
Brendan the Vending Machine	
Turn-in directory : *ex*02/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `n/a`	

Setup a vending machine to gain more profit!

- Create the `Vendingmachine` class as follows inside vendingmachine.py:

```
class Vendingmachine:
    def __init__(self, name, stock):
        """
        taking a string, a list of Beverage instances as parameters,
        assign its value to a name attribute and a stock attribute.
        """
        pass

    def __str__(self):
        """
        return its name attributes in the format.
        <name attribute> the vending machine
        """
        pass

    def sell(self, beverage_name):
        """
        taking a string as a parameter, print a message of your choice.
        the message differ when there is a Beverage instance
        named same as beverage_name in the stock, or not.
        """
        pass

    def ask(self):
        """
        print a message to ask the beverage name,
        store the user input,
        then call sell(beverage_name) function.
        """
        pass
```

for example, the `ask` method's output can be like this:

- for example, the `ask` method's output can be like this:

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

- or this:

```
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 03 : Sell them more.

Setup a better UX to gain more profit!

	Exercise 03
better UX	
Turn-in directory : *ex*03/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `random`	

- Create the `Vendingmachine` class as follows inside vendingmachine.py:

```
class Vendingmachine:
    def __init__(self, name, greeting, stock):
        """
        taking 2 string, a list of Beverage instances as parameters,
        assign its value to a name attribute, a greeting attribute and a stock attribute.
        """
        pass

    def greet(self):
        """
        print a greeting attribute of the instance.
        """
        pass

    def display(self):
        """
        show its lineup of stocks.
        """
        pass

    def recommend(self):
        """
        choose a random instance from its stock,
        then print a recommendation sentence using the name attribute of the selected Beverage
             instance.
        """
        pass
```

- add ___str___ method from the previous exercise.

```
    def __str__(self):
        """
        return its name attributes in the format.
        <name attribute> the vending machine
        """
        pass
```

- add ask method from the previous exercise.

```
    def ask(self):
        """
        call display method and print a message to ask the beverage name,
        store the user input,
        then call sell(beverage_name) function.
        """
        pass
```

- rewrite sell method as follows:

```
    def sell(self, beverage_name):
        """
        print a message of your choice.
        the message differ when there is a Beverage instance
        named same as beverage_name in the stock, or not
        call recommend method
        when the given beverage name is not found in its stock.
        """
        pass
```

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

```
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 04 : Can't be infinite.

	Exercise 04
finite stock	
Turn-in directory : *ex*04/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `random`	

The Vending machine doesn't check stocks. We need to fix this as soon as possible! Add or rewrite `Vendingmachine` class as follows:

- `__init__` now takes a list of dictionaries for the stock attribute. Each dictionary contains two key-value pairs:
  - 'product': a Beverage instance
  - 'amount': integer (indicates how many drinks are in stock)

```
class Vendingmachine:
    def __init__(self, name, greeting, stock):
        """
        taking 2 string, a list of dictionaries {'product': a Beverage instance, 'amount':
            integer} as parameters,
        assign its value to a name attribute, a greeting attribute and a stock attribute.
        """
        pass
```

  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", [{'product':coca,'amount':1},{'product':coffee,'
    amount':5}])
```

- add all the methods in the previous exercise.
- rewrite `A sell` method from the previous exercise, so that it sells the beverage if there is any stock left and it reduce the stock when it is sold.
- `A add_stock` method.

```
    def add_stock(self, beverage_name, number):
        """
        taking a string and integer as parameters,
        increase the amount of that drink in stock.
        when the Beverage instance with the same name as beverage_name exists in the stock.
        if not, create a new Beverage instance and add it into its stock with the number
        """
        pass
```

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 05 : I am your manager.

	Exercise 05
mangers manage their stock	
Turn-in directory : *ex*05/	
Files to turn in : `beverage.py, vendingmachine.py, main.py`	
Allowed functions : `random`	

We need to setup admin mode to be able to fill the stocks.
Add or rewrite the `Vendingmachine` class as follows:

- `__init__` taking another string as a parameter in addition to all the previous parameters. This parameter should be assigned to a admin_code attribute.

```
  class Vendingmachine:
      def __init__(self, name, greeting, stock, admin_code):
          """
          taking 3 string, a list of dictionaries {'product': a Beverage instance, 'amount':
              integer} as parameters,
          assign its value to a name attribute, a greeting attribute a stock attribute, and
              admin_code attribute.
          """
          pass
```

  For example,

```
vm = Vendingmachine("Henry", "Hi, I am Henry", [{'product':coca,'amount':1},{'product':coffee,'
    amount':5}], "I am your admin.")
```

- add all the methods in the previous exercise.

- `ask` method from the previous exercise, yet inputting a string "admin mode" instead of beverage name will not sell a beverage but start its admin mode, calling a method admin_mode.

```
def ask(self):
        """
        call display method and print a message to ask the beverage name,
        store the user input,
        then call sell(beverage_name) function.
        if a string "admin mode" is inputted instead of beverage name,
        start its admin mode, calling a method admin_mode.
        """
        pass
```

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

```
def show_stock(self):
"""
displays all the stocked beverage with each number of stock in the form of your choice.
"""
pass
```

- `admin_mode` method. Implement this method by mimicing the behavior from the example and the docstring. You are free to create your own sentence.

```
def admin_mode(self):
    """
    prompt 'Please tell me the admin code:' to the user, wait for the user's input
    and exit if the user's input doesn't match the admin_code attribute.
    If the user's input match with admin_code attribute,
    prompt a name of beverage and the number of stock to add.
    Then, call the previously implemented add_stock method.
    If "exit" is inputted, then it will end admin_mode.
    If "show stock" is inputted instead of a beverage name,
    this method will call another method show_stock instead.
    """
    pass
```

Example of Henry the vending machine's admin mode:

```
would you like something?
> admin mode

Please tell me the admin code:
> I am your admin.

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

Do you want to show stock or add a stock? Type "show stock" to show my stock or a beverage name
     to add stock.
> coca-cola

How many stock do you want to add?
> 5

Done.

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! ---> 10 left

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

How many stock do you want to add?
> 5

Done.
```

```
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! ---> 10 left
AAAAA : no description ---> 5 left

Do you want to show stock or add a stock? Type "show stock" to show my stock or a beverage name
     to add stock.
> coca-cola

How many stock do you want to add?
> A

Not a valid input.

Do you want to show stock or add a stock? Type "show stock" to show my stock or a beverage name
     to add stock.
> exit
Bye !
```

- 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 06 : Bonus

	Exercise 06
The more original, the better	
Turn-in directory : *ex*06/	
Files to turn in : `*`	
Allowed functions : `n/a`	

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

- Import previously created `beverage.py` and `product.py`.
- unittest (https://docs.python.org/ja/3/library/unittest.html)
- `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.
- Implement a feature which 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 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.