OBJECT-ORIENTED PYTHON PRACTICE PROBLEMS

Contents

Tên	MSSV	Phần việc
Trần Nguyễn Quang Lâm	20139040	3 câu đầu
Tô Gia Huy	20139003	3 câu cuối

1. Person Class

In this practice problem, you'll create a class to represent a person. For simplicity, we will only store the first and last names of a person
We want you to implement a class named `Person` with the following methods:

- `__init__(self, first_name, last_name)`: This method creates a `Person` instance by storing `first_name` into `self.first_name` and `last_name` into `self.last_name`

- `__str__(self, other)`: This method computes a string representation of this person. The format of this string should be the first name, followed by a space, and then the last name. In each name, we want all characters to be in lower case, except for the first one that should be in upper case.
- `__init__(self, first_name, last_name):` method lưu trữ tên của một người, đối số first_name truyền vào là tên, last_name là họ
- `__str__(self, other):` method định dạng chuẩn lại input tên của người được nhập vào. Định dạng chuẩn ở đây có thể được hiểu là ghi hoa chữ cái đầu tiên, các chữ cái còn lại không ghi hoa, trong tên không có khoảng trắng

Examples of usage:

```
person = Person('bruno', 'LopeZ')
print(person)
```

```
Bruno Lopez
```

```
person = Person('aNNa', 'martin')
print(person)
```

```
Anna Martin
```

1. Define a class named `Person`.
2. Define the `__init__()` method with three arguments:
   - `self` : The self-reference of the class instance.
   - `first_name` : The first name of the person.
   - `last_name` : The last name of the person.
3. Implement the `__init__()` method so that it stores `first_name` in `self.first_name` and `last_name` in `self.last_name`.
4. Define a `__str__()` method with one arguments:
   - `self` : The self-reference of the class instance.
5. Implement the `__str__()` method so that it returns a string representation of this person. The format of this string should be the first name, followed by a space, and then the last name. In each name, we want all characters to be in lower case, except for the first one that should be in upper case.

Các bước thiết lập class và method:

+Khai báo class Person, đây là class chứa method để định dạng chuẩn tên

+Định nghĩa method __init__() với 3 đối số truyền vào:

+self: tự tham chiếu đến biến được gọi

+first_name:tên của người được nhập

+last_name: họ của người được nhập

+Thiết lập, triển khai method __init__, lưu tên vào self.first_name

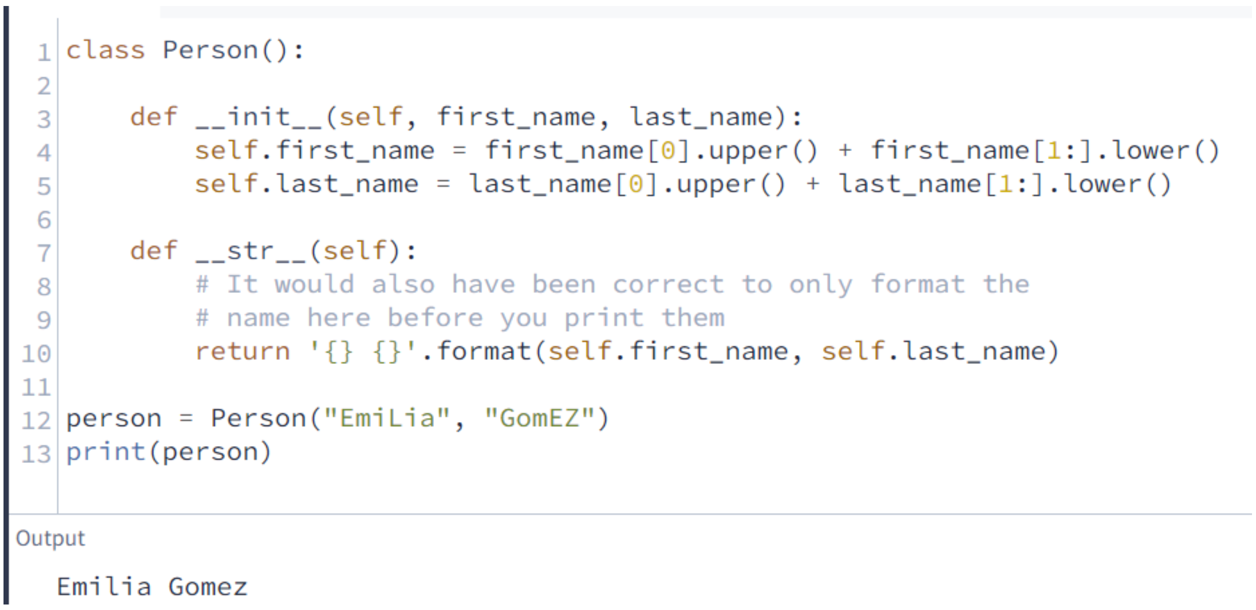
+Định nghĩa method __str()__ với một đối số truyền vào:

+self: tự tham chiếu đến biến được gọi

+Thiết lập và triển khai method __str__ để nó trả về định dạng chuẩn của tên người đã nhập

Optional steps to test your solution:

1. Create an instance of `Person` using `"EmiLia"` for the first name and `"GomEZ"` as the last name and assign it to a variable named `person`.
2. Print the value of `person`. The result should be `Emilia Gomez`.



```python
class Person():

    def __init__(self, first_name, last_name):
        self.first_name = first_name[0].upper() + first_name[1:].lower()
        self.last_name = last_name[0].upper() + last_name[1:].lower()

    def __str__(self):
        # It would also have been correct to only format the
        # name here before you print them
        return '{} {}'.format(self.first_name, self.last_name)

person = Person("EmiLia", "GomEZ")
print(person)
```

Output

```
Emilia Gomez
```

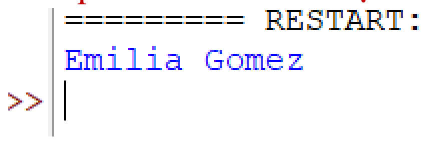
->Class Person với 2 method

+__init__() In hoa chữ cái đầu, các chữ cái sau ghi thường và lưu tên của người nhập vào self.first_name, self.last_name

+__str__ Trả về kết quả tên chuẩn hóa

Khởi tạo biến person và gán vào class Person, truyền vào 2 đối số là tên và họ Emilia, Gomez

Kết quả trả về là tên được chuẩn hóa Emilia Gomez

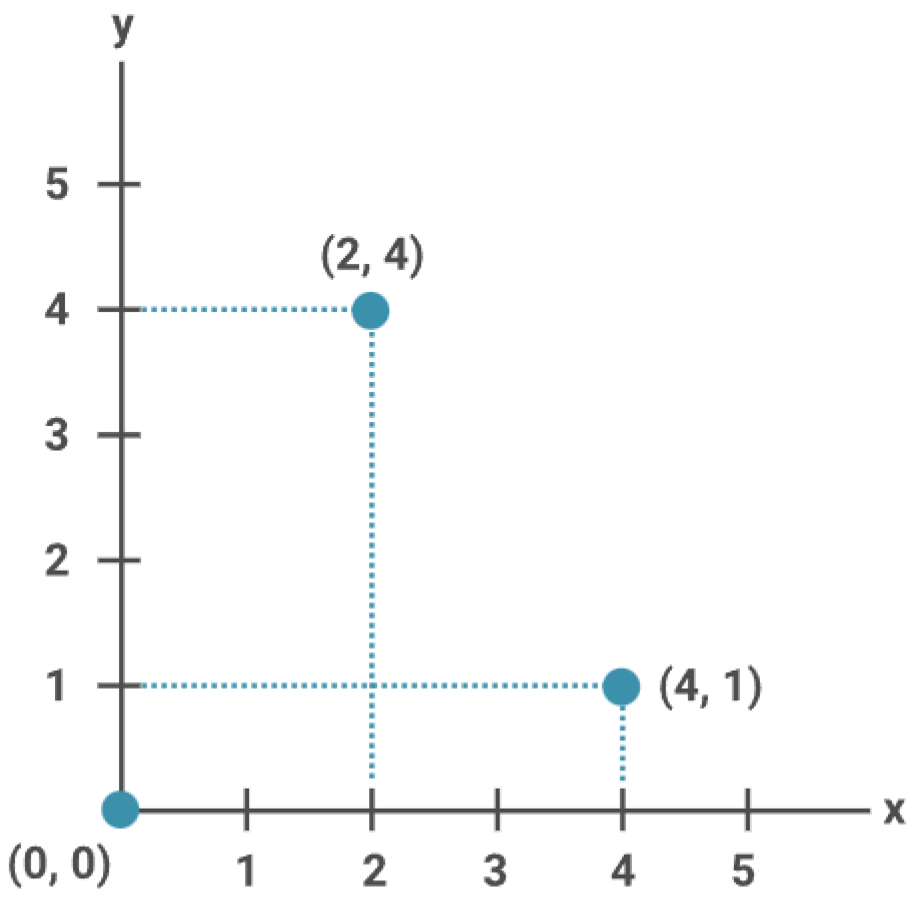


```
========= RESTART:
Emilia Gomez
>> |
```

2. 2D Points

In this practice problem, you'll create a class to represent points in two dimensions. /Tạo một class để hiển thị các điểm trong không gian 2 chiều/

A point in two dimensions is essentially a pair of numbers.
The following figure shows points *(0, 0)*, *(2, 4)* and *(4, 1)*:



We want you to implement a class named `Point2D` with the following methods:

- `__init__(self, x, y)` : This method creates a `Point2D` instance by storing `x` into `self.x` and `y` into `self.y` .
- `distance(self, other)` : This method computes the distance between points `self` and `other` .

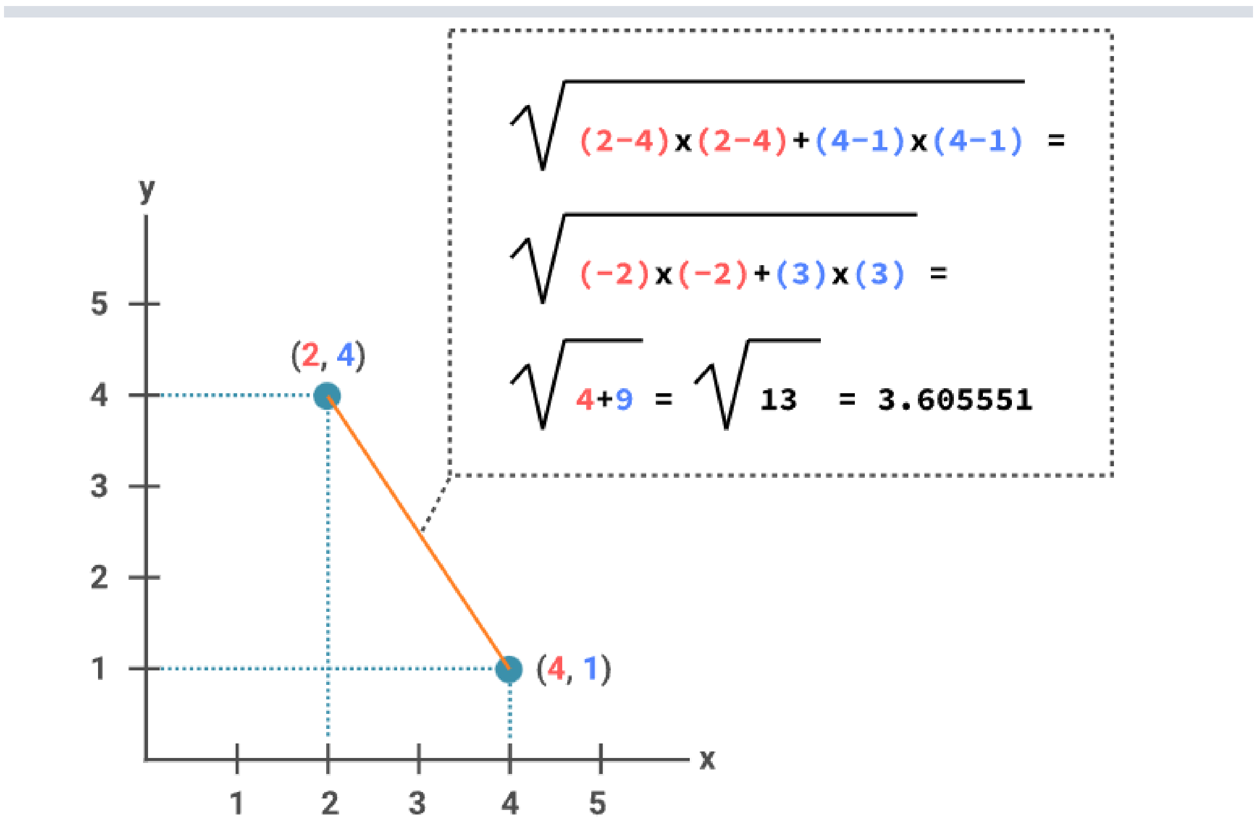
Yêu cầu khởi tạo class tên Point2D với các method:

+__init__(self, x, y): Tạo ra một biến Point2D, lưu vị trí x vào self.x và y vào self.y

+distance(self, other): tính khoảng cách giữa self(điểm đã cho) và other(điểm khác) với công thức như hình dưới hoặc dùng hàm math.sqrt() từ class math

The distance between two points *(x_1, y_1)* and *(x_2, y_2)* is calculated with the following formula

For example, the distance between points *(2, 4)* and *(4, 1)* is approximatively equal to 3.605551, as shown in the figure:



In Python, you can compute the square root of a number by using the `math.sqrt()` function from `math` module. Here are a few examples:

```
import math
print(math.sqrt(9))
print(math.sqrt(1))
print(math.sqrt(42))
```

```
3.0
1.0
6.48074069840786
```

Here is an example of how someone might use your class once implemented:

```
point1 = Point2D(2, 4)
point2 = Point2D(4, 1)
distance = point1.calculate_distance(point2)
print(distance)
```

```
3.605551275463989
```

1. Define a class named `Point2D`.
2. Define the `__init__()` method with three arguments:
   - `self` : The self-reference of the class instance.
   - `x` : The value of the x-coordinate.
   - `y` : The value of the y-coordinate.
3. Implement the `__init__()` method so that it stores `x` in `self.x` and `y` in `self.y`.
4. Define a `calculate_distance()` method with two arguments:
   - `self` : The self-reference of the class instance.
   - `other` : Another instance of `Point2D` to which we want to compute the distance.
5. Implement the `calculate_distance()` method so that it returns the distance between this point and the one given as argument.

->Các bước triển khai class:

+)Khai báo class Point2D

+)Khai báo method __init__() với 3 đối số truyền vào:

-self: tự tham chiếu đến biến được khai báo thuộc class Point2D này

-x: tọa độ x

-y: tọa độ y

+)Triển khai method __init_() với các dòng lệnh lưu tọa độ x vào self.x(thuộc tính tọa độ x của biến đã khai báo thuộc class Point2D

+)Khai báo method calculate_distance() để tính khoảng cách với 2 đối số truyền vào

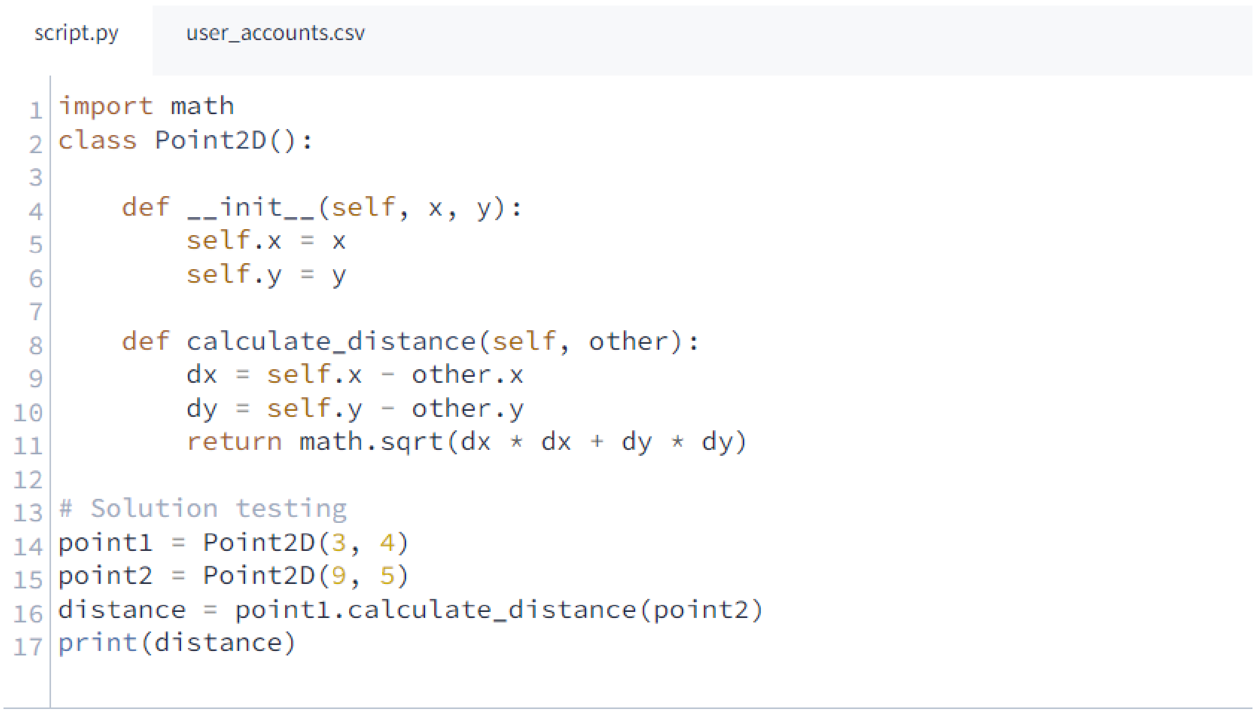
-self: tự tham chiếu đến biến đã được khai báo thuộc class Point2D

-other: biến khác trong cùng class Point2D để tính khoảng cách)

+)Triển khai tính toán trong method này

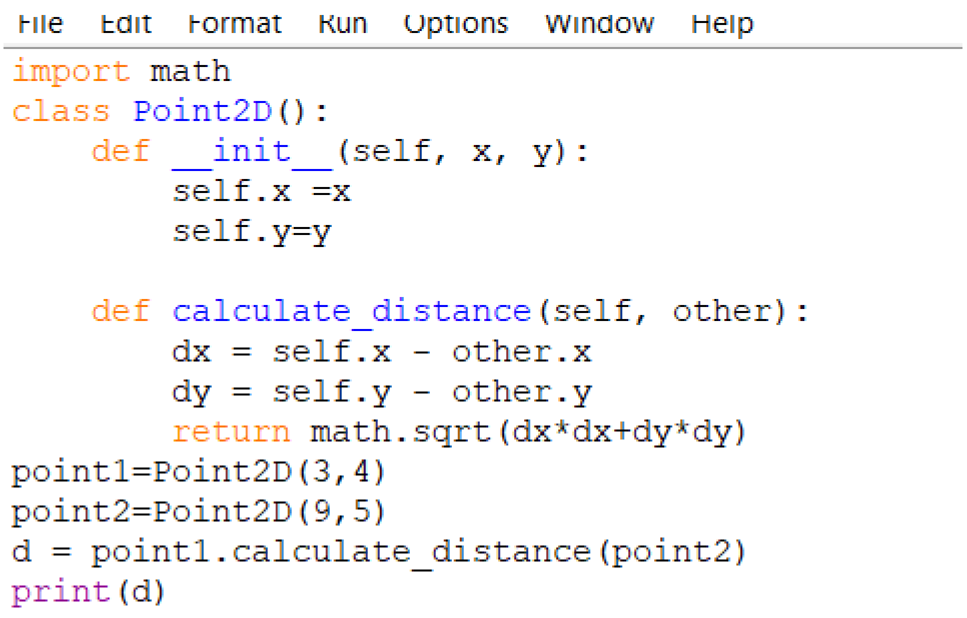
Optional steps to test your solution:

1. Create an instance of `Point2D` to represent point *(3, 4)* and assign it to variable `point1`.
2. Create an instance of `Point2D` to represent point *(9, 5)* and assign it to variable `point2`.
3. Test your class by calculating the distance between `point1` and `point2` and assign the result to a variable named `distance`.
4. Print the value of `distance`. The result should be approximately equal to `6.082762530298219`.



```python
import math
class Point2D():

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def calculate_distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        return math.sqrt(dx * dx + dy * dy)

# Solution testing
point1 = Point2D(3, 4)
point2 = Point2D(9, 5)
distance = point1.calculate_distance(point2)
print(distance)
```

Code:



```python
import math
class Point2D():
    def __init__(self, x, y):
        self.x =x
        self.y=y

    def calculate_distance(self, other):
        dx = self.x - other.x
        dy = self.y - other.y
        return math.sqrt(dx*dx+dy*dy)
point1=Point2D(3,4)
point2=Point2D(9,5)
d = point1.calculate_distance(point2)
print(d)
```

KQ

```
>>> 
    ========== RESTART: C:/
    6.082762530298219
>>> |
```

3. Time

In this practice problem, you'll be creating a class to represent a time of the day. We represent the time of the day with three values: hour, minute, and second.
For example, midday has hour equal to 12 and both minute and second equal to 0: `12:00:00` (the first two digits represent the hour, the next two the minute and the last two the seconds). The last second of a day will be `23:59:59.`
`->Tạo ra class để hiển thị thời gian thực với định dạng giờ:phút:giây`

We want you to implement a class named `Time` with the following methods:

- `__init__(self, hour, minute, second)` : This method creates a `Time` instance by storing `hour` into `self.hour` , `minute` into `self.minute` and `second` into `self.second` . These three values will be integers representing the time hour, minute, and second, respectively.

Example of usage:

```
time1 = Time(12, 0, 0)
time2 = Time(23, 59, 59)
time3 = Time(9, 15, 37)
print(time3.hour, time3.minute, time3.second)
```

```
9 15 37
```

-> __init__(self, hour,minute,second): method tạo ra biến thời gian, lưu giá trị giờ vào self.hour (thuộc tính giá trị giờ của biến tạo ra thuộc class này), giá trị phút vào self.minute, giây vào self.second

- `total_seconds(self)` : Returns the number of seconds that this time corresponds to by multiplying the hour by 3,600, the minute by 60, and adding the results together with the second.

Example of usage:

```
time1 = Time(0, 0, 0)
time2 = Time(0, 0, 1)
time3 = Time(0, 1, 1)
time4 = Time(1, 1, 1)
print(time1.total_seconds())
print(time2.total_seconds())
print(time3.total_seconds())
print(time4.total_seconds())
```

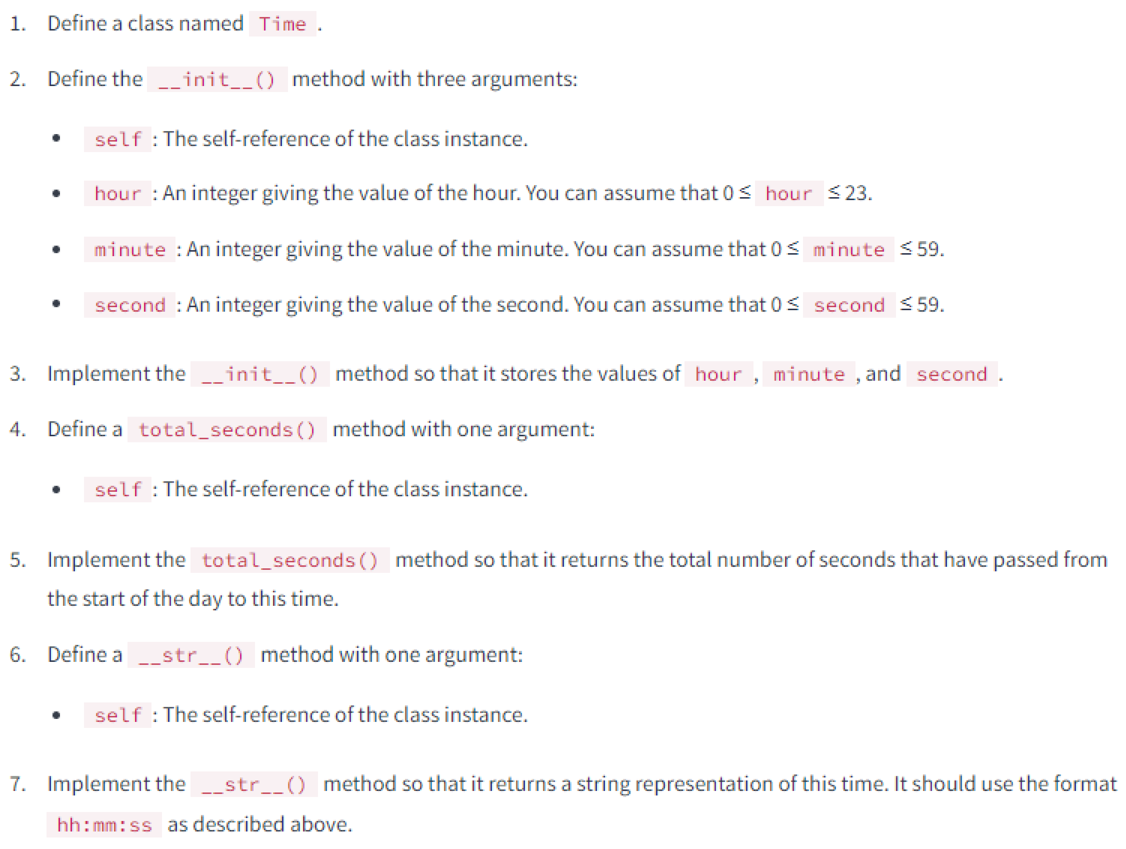
```
0
1
61
3661
```

- `__str__(self)` : Returns a string representation of the time so that when we print it, it looks good. The string should have the format `hh:mm:ss` . That is, use two digits to represent the hour, minute, and second adding a 0 in front if necessary.

Example of usage:

```
time1 = Time(12, 0, 0)
time2 = Time(23, 59, 59)
time3 = Time(9, 15, 37)
print(time1)
print(time2)
print(time3)
```

```
12:00:00
23:59:59
09:15:37
```



1. Define a class named `Time` .

2. Define the `__init__()` method with three arguments:

   - `self` : The self-reference of the class instance.
   - `hour` : An integer giving the value of the hour. You can assume that 0 ≤ `hour` ≤ 23.
   - `minute` : An integer giving the value of the minute. You can assume that 0 ≤ `minute` ≤ 59.
   - `second` : An integer giving the value of the second. You can assume that 0 ≤ `second` ≤ 59.

3. Implement the `__init__()` method so that it stores the values of `hour` , `minute` , and `second` .

4. Define a `total_seconds()` method with one argument:

   - `self` : The self-reference of the class instance.

5. Implement the `total_seconds()` method so that it returns the total number of seconds that have passed from the start of the day to this time.

6. Define a `__str__()` method with one argument:

   - `self` : The self-reference of the class instance.

7. Implement the `__str__()` method so that it returns a string representation of this time. It should use the format `hh:mm:ss` as described above.

->Các bước triển khai class Time:

+)Khai báo class Time:

+)Khai báo method __init__() với 4 đối số :

-self: tự tham chiếu đến biến đã tạo thuộc class Time

-hour: giá trị nguyên chỉ giờ của biến , cho điều kiện 0 <= hour <=23

-minute: giá trị nguyen chỉ phút, 0<=minute<=59

-second: giá trị nguyên chỉ giây, 0<= second<=59

+) Triển khai method bằng các dòng lệnh lưu các giá trị giờ phút giây vào biến đã khai báo

+)Khai báo method total_seconds() với một đối số self: tự tham chiếu đến biến được khai báo

+)Triển khai method để trả về giá trị số giây từ đầu ngày đến thời gian method này được gọi

+)Khai báo method __str__() một đối số truyền vào self: tự tham chiếu đến biến đã khai báo thuộc class Time

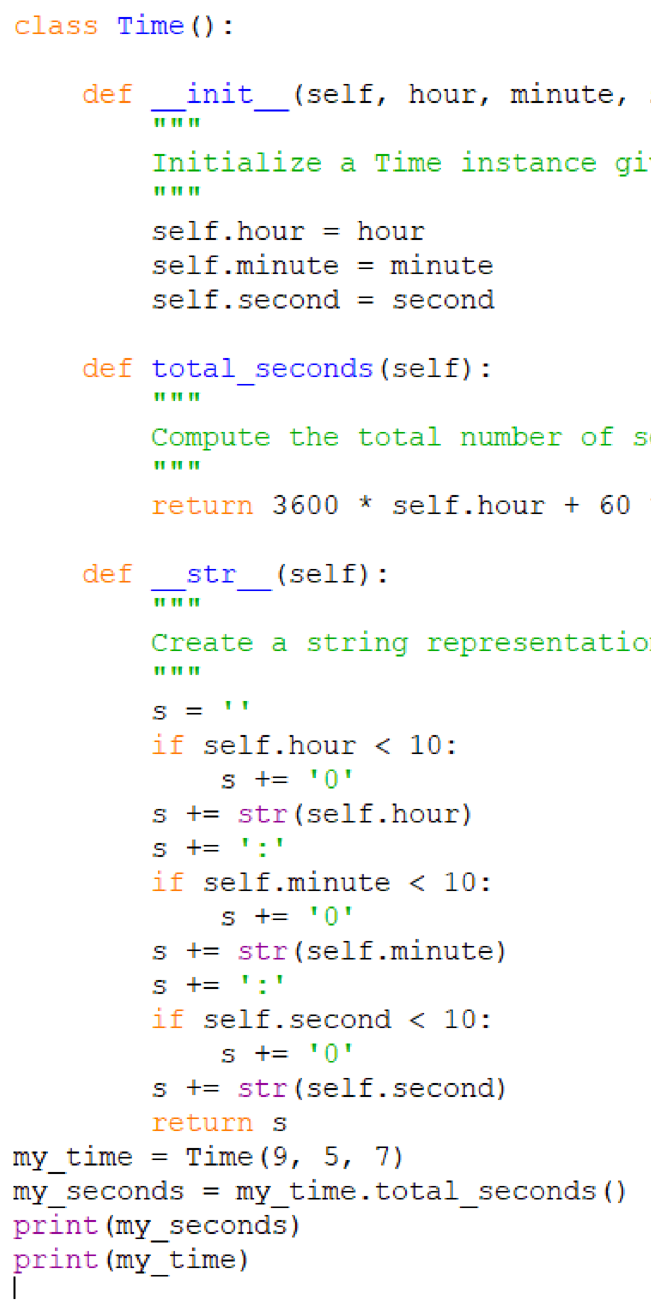
+)Triển khai method __str__() để trả về thời gian dạng chuẩn

Optional steps to test your solution:

1. Create an instance of `Time` to represent the time `09:05:07` and assign it to variable `my_time`.
2. Using the `total_seconds()` method, calculate the total number of seconds in this time. Assign the result to a variable named `my_seconds`.
3. Print the value of `my_seconds`. The result should be equal to `32707`.
4. Print the value of `my_time`. The result should be equal to `09:05:07`.

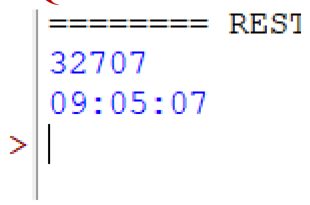
```
class Time():

    def __init__(self, hour, minute, second):
        """
        Initialize a Time instance giving the hour, minute and second.
        """
        self.hour = hour
        self.minute = minute
        self.second = second

    def total_seconds(self):
        """
        Compute the total number of seconds since the start of the day.
        """
        return 3600 * self.hour + 60 * self.minute + self.second

    def __str__(self):
        """
        Create a string representation of this time.
        """
        s = ''
        if self.hour < 10:
            s += '0'
        s += str(self.hour)
        s += ':'
        if self.minute < 10:
            s += '0'
        s += str(self.minute)
        s += ':'
        if self.second < 10:
            s += '0'
        s += str(self.second)
        return s

# Solution testing
my_time = Time(9, 5, 7)
my_seconds = my_time.total_seconds()
print(my_seconds)
print(my_time)
```

Code:



```
class Time():

    def __init__(self, hour, minute, second):
        """
        Initialize a Time instance giving the hour, minute and second.
        """
        self.hour = hour
        self.minute = minute
        self.second = second

    def total_seconds(self):
        """
        Compute the total number of seconds since the start of the day.
        """
        return 3600 * self.hour + 60 * self.minute + self.second

    def __str__(self):
        """
        Create a string representation of this time.
        """
        s = ''
        if self.hour < 10:
            s += '0'
        s += str(self.hour)
        s += ':'
        if self.minute < 10:
            s += '0'
        s += str(self.minute)
        s += ':'
        if self.second < 10:
            s += '0'
        s += str(self.second)
        return s
my_time = Time(9, 5, 7)
my_seconds = my_time.total_seconds()
print(my_seconds)
print(my_time)
```

KQ:



```
========= REST
32707
09:05:07
>
```

4. Frequency Table

In this practice problem, you'll create a FreqTable class to implement a frequency table data structure.
>>Tạo 1 class Freqtable để thực hiện lưu tần số trong bảng

This data structure will initially start empty and then allow users to add elements using a specific method. Every time we add an element, we increase its count. Then we can retrieve the total number of occurrences of an element using another method.

Here is an example of how one would use it:

```
freq_table = FreqTable()
freq_table.add('a') # Add element 'a'
freq_table.add('a') # Add element 'a'
freq_table.add('b') # Add element 'b'
print(freq_table.get_count('a')) # Get how many 'a' there is
print(freq_table.get_count('b')) # Get how many 'b' there is
print(freq_table.get_count(100)) # Get how many 100 there is
```

```
2 # Number of 'a'
1 # Number of 'b'
0 # Number of 100
```

Implement a `FreqTable` class with the following methods:

- `__init__(self)` : This method creates a `FreqTable` instance. Initially, the frequency table is empty, so the only thing that this method does is creating and storing an empty dictionary into `self.count` .
- `add(self, element)` : This method increases the count of `element` by 1.
- `get_count(self, element)` : This method returns the count of `element` . If the `element` was never added, it should return 0.

- _init__(self) method tạo ra 1 biến FreqTable. Ngay lúc đầu, bảng tần số rỗng, nên thứ duy nhất method này tạo ra là self.count rỗng.
- add(self,element) tăng count của phần tử được thêm lên 1.
- get_count(self, element) trả về số lượng của phần tử, nếu phần tử đó chưa được thêm hoặc chưa khởi tạo, sẽ đều trả về 0.

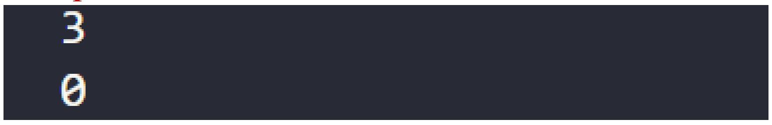
1. Define a class named `FreqTable`.
2. Define the `__init__()` method with three arguments:
    - `self`: The self-reference of the class instance.
3. Implement the `__init__()` method so that it creates an empty dictionary and assigns is to `self.count`.
4. Define a `add()` method with two arguments:
    - `self`: The self-reference of the class instance.
    - `element`: The element that we want to increase the count of. You can assume that `element` is hashable. This means that it can be used as a key in a dictionary.
5. Implement the `add()` method so that it increases the count of `element` by one. It should return the current count of the `element` that was just added.
6. Define a `get_count()` method with two arguments:
    - `self`: The self-reference of the class instance.
    - `element`: The element that we want to know the count of. You can assume that `element` is hashable. This means that it can be used as a key in a dictionary.
7. Implement the `get_count()` method so that it returns the number of occurrences of `element`. Return 0 is `element` was never added.

Optional steps to test your solution:

1. Create an instance of `FreqTable` and assign it to `freq_table`.
2. Using the `add()` method, add value `0` three times into `freq_table`.
3. Using the `get_count()` method, print the count of `0` and the count of `1`.

```python
class FreqTable():

    def __init__(self):
        self.count = {}

    def add(self, element):
        # Check if this is the first time
        if not element in self.count:
            self.count[element] = 0
        self.count[element] += 1
        return self.count[element]

    def get_count(self, element):
        # Check if the element was ever added
        if element not in self.count:
            return 0
        return self.count[element]

# Solution testing
freq_table = FreqTable()
for _ in range(3):
    freq_table.add(0)
print(freq_table.get_count(0))
print(freq_table.get_count(1))
```

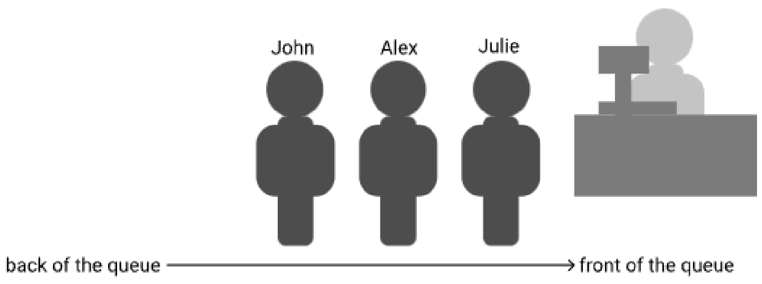
>>Khởi tạo self.count = {}(dictionary)
>>Method add: khi ta add 1 phần tử, kiểm tra xem phần tử đó đã tồn tại trong {} hay chưa, nếu chưa, khởi tạo count của object đó bằng 0, sau đó sẽ cộng count của object đó thêm 1.
>>Method get_count: để trả về số lượng phần tử đó trong dictionary.
Kết quả của code trên:



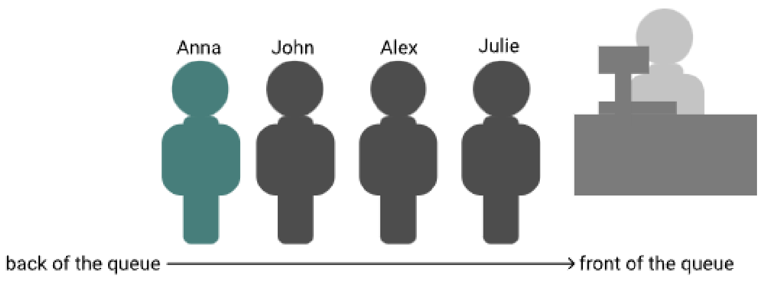
```
3
0
```

5. Supermarket Queue

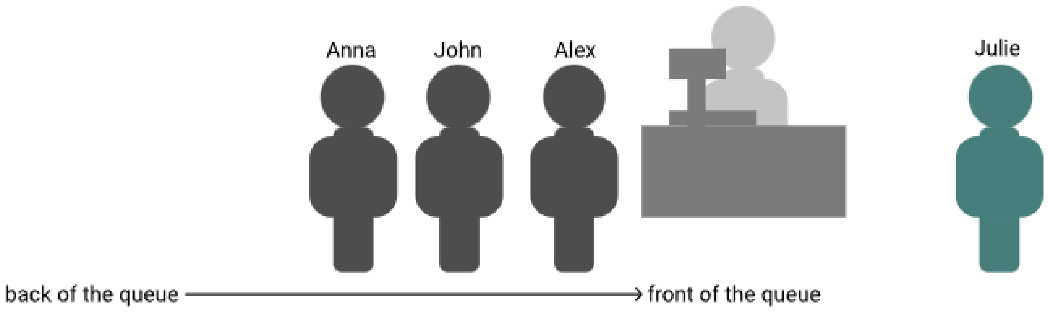
In this practice problem, you'll create a class to model a supermarket queue. We will want to add people to the back of the queue and remove them from the front of the queue as they leave the supermarket.
>> Tạo ra 1 model hàng đợi siêu thị. chúng ta sẽ muốn mọi người trở về hàng đợi và đưa họ ra hàng đợi như là họ rời khỏi siêu thị.
For example, imagine that we have a queue with three people, as shown below:
Giả sử hàng đợi có 3 người



When Anna arrives at the queue, she will go to the back of the queue:



>>Anna thêm vào hàng, sau khi julie xong, người tiếp theo sẽ là Alex, đương nhiên julie sẽ ra khỏi hàng
Julie will be the first to be served as she is in front of the queue. When she is done, Anna, John, and Alex will remain in the queue. The next person to be served will be Alex:



>>Julie ra khỏi hàng

For simplicity, we will use strings with the people's names to represent the people in the queue. We want you to implement a class named `SupermarketQueue` with the following methods:

- `__init__(self)` : This method creates a `SupermarketQueue` instance. The queue will initially be empty.

Example of usage:

```
queue = Queue()
```

- `add_to_back(self, name)` : Adds a person named `name` to the back of the queue.

Example of usage:

```
queue.add_to_back('Julie')
queue.add_to_back('Alex')
queue.add_to_back('John')
```

- `remove_from_front(self)` : Removes the person in front of the queue. It should return that person's name. You can assume that this method will never be called if the queue is empty.

Example of usage:

```
removed = queue.remove_from_front()
print(removed)
```

```
Julie
```

- `__str__(self)` : Returns a string representation of the queue. The format should be the same that lists use. That is, square brackets `[]` with the element separated by a comma and a space `, `. The elements should be in a back to front order. This means that the first is the one in the back of the queue, and the last is the one in the front.

Example of usage:

```
print(queue)
```

```
['John', 'Alex']
```

- `__len__(self)` : Returns the number of people in the queue.

Example of usage:

```
print(len(queue))
```

```
2
```

There are several possible ways to implement a queue internally. We will let you decide how you implement yours as long as your methods comply with the provided specification.

1. Define a class named `SupermarketQueue`.
2. Define the `__init__()` method with one argument:
   - `self` : The self-reference of the class instance.
3. Implement the `__init__()` method so that it creates an empty queue.
4. Define a `add_to_back()` method with two arguments:
   - `self` : The self-reference of the class instance.
   - `element` : Add an element at the back of the queue.
5. Implement the `add()` method so that add the given element to the back of the queue.
6. Define a `remove_from_front()` method with one argument:
   - `self` : The self-reference of the class instance.
7. Implement the `remove_from_front()` that removes and returns the element in the front of the queue.
8. Define a `__str__()` method with one argument:
   - `self` : The self-reference of the class instance.

9. Implement the `__str__()` method so that it returns a string representation of the current state of the queue. It should use the same format as lists do. That is, square brackets `[]` with the element separated by a comma and a space `, `. For example, if the queue contains John and Alex and Alex if at the front, then your method should return `'[John, Alex]'`.

10. Define a `__len__()` method with one argument:

    - `self` : The self-reference of the class instance.

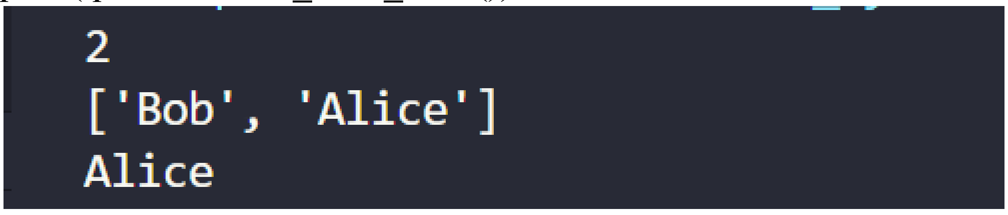
11. Implement the `__len__()` method so that it returns the number of elements currently in the queue.

Optional steps to test your solution:

1. Create an instance of `SupermarketQueue` and assign it to `queue`.
2. Using the `add_to_back()` method, add value `'Alice'`.
3. Using the `add_to_back()` method, add value `'Bob'`.
4. Print the length of the queue. The result should be `2`.
5. Print the queue. The result should be `['Bob', 'Alice']`.
6. Print the result of `remove_from_front()`. The result should be `Alice`.

1. Tạo 1 biến thuộc SupermarketQueue và giao nó cho queue
2. Dùng method add_to_back(), thêm Alice
3. Tương tự bước 2, thêm Bob
4. In ra độ dài hàng đợi
5. In ra hàng đợi có ai.
6. In ra kết quả, ai sẽ là người đi ra hàng đợi đầu tiên

```
class SupermarketQueue():

    def __init__(self): #method init để khởi tạo các nhân tố cần.
        self.elements = []
        self.front_index = 0
        self.num_elements = 0

    def add_to_back(self, element): # khi thêm, ta sẽ thêm vào đuôi của hàng đợi
        self.elements.append(element)
        self.num_elements += 1

    def remove_from_front(self):  # xóa phần tử vào trước( đầu hàng )
        ret = self.elements[self.front_index]
        self.front_index += 1
        self.num_elements -= 1
        return ret

    def __str__(self):              #dùng để ptrint string elements
        tmp = self.elements[self.front_index:].copy()
        tmp.reverse()
        return str(tmp)

    def __len__(self):   #độ dài của elements
        return self.num_elements

# Solution testing
queue = SupermarketQueue()
queue.add_to_back('Alice')
queue.add_to_back('Bob')
print(len(queue))
print(queue)
print(queue.remove_from_front())
```



```
2
['Bob', 'Alice']
Alice
```

6. Using a Class for CSV Rows

When dealing with data from a CSV file as a list of lists, it can sometimes make the code hard to read. What makes the code hard to read is that, for a given row, we use row[0], row[1] and, so on, to refer to the values. The code would be much more readable if we could use the column names.
>> code sẽ dễ đọc hơn khi ta sử dụng các tên của cột.
Let's consider the user_accounts.csv file. This file was randomly generated for this practice problem. It does not contain real data. Here are the first five rows:
>>Dòng đầu tiên sẽ không chứa dữ liệu thực.

id	email	name	address
0	anna.carter@gmail.com	Anna Carter	27183 Craig Shore Suite 886 New Benjamin TN 92858
1	joseph.kirby@yahoo.com	Joseph Kirby	3594 Fox Ford Apt. 192 West Kristen GA 22838-8977
2	larry.cain@martinez.net	Larry Cain	58208 Cook Bypass West Benjaminfurt OH 25179
3	johnathan.soto@yahoo.com	Johnathan Soto	68536 Avery Expressway Amberton PA 95197
4	paula.acosta@yahoo.com	Paula Acosta	065 Kayla Alley Apt. 098 Walkerview NH 14274

To make it easier to handle the data from this CSV file, you'll create a `User` class with one field for each column in the CSV file. In this way, one will be able to use these fields to refer to the values that represent a user rather than using indexes.
>>Dùng User class với mỗi cột cho file CSV. Với cách này, ta có thể sử dụng tên của các cột thay vì dùng index của các cột để truy xuất dữ liệu

We want you to implement a class named `User` with the following methods:

- `__init__(self, row)` : This method creates a `User` instance by assigning `row[0]` to `self.id` , `row[1]` to `self.email` , `row[2]` to `self.name` and `row[3]` to `self.address` .

Examples of usage:

```
user = User(['0', 'anna.carter@gmail.com', 'Anna Carter', '27183
Craig Shore Suite 886 New Benjamin TN 92858'])
print(user.id)
print(user.email)
print(user.name)
print(user.address)
```

>>Chúng ta muốn tạo ra 1 class User với:

- row[0] chứa sefl.id
- row[1] chứa self.email
- row[2] self.name
- row[3] self.address

```
0
anna.carter@gmail.com
Anna Carter
27183 Craig Shore Suite 886 New Benjamin TN 92858
```

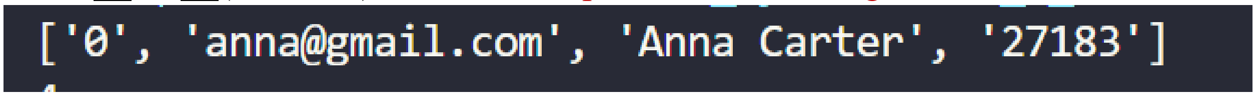
1. Define a class named `User`.
2. Define the `__init__()` method with two arguments:
   - `self`: The self-reference of the class instance.
   - `row`: A row from the CSV. You can assume it contains four elements. The first will be the user id, the second the user email, the third the user name, and the forth the user address.
3. Implement the `__init__()` method so that it stores `row[0]` into `self.id`, `row[1]` into `self.email`, `row[2]` into `self.name` and `row[3]` into `self.address`.
4. Read the `user_accounts.csv` file.
5. Create a list named `users`. Populate it with instances of `User`, one for each row in the CSV file. Remember to ignore the first row, which contains the column headers.

1. Định nghĩa class User
2. Định nghĩa method __init__ với 2 biến: self và row(theo em hiểu thì row ở đây biểu diễn cho cột)
3. Thực hiện method __init__
4. Đọc file csv
5. Tạo mảng users. lưu các hàng là của Class User trả về

Optional steps to test your solution:

1. Assign the last user to a variable named `last`.
2. Print the value of `last.id`. You should get `9999`.
3. Print the value of `last.email`. You should get `edward.jimenez@lee.org`.
4. Print the value of `last.name`. You should get `Edward Jimenez`.
5. Print the value of `last.address`. You should get `4445 Dylan Brook Apt. 099 South Lisa DE 87826`.

```
# User class
class User():

    def __init__(self, row):  # row được put vào là cả 1 hàng của file VD
```



```
['0', 'anna@gmail.com', 'Anna Carter', '27183']
```

(row ở method này sẽ chứa dữ liệu như ảnh)

```
        self.id         = row[0]   #  index sau row ở đây thực ra là index của cột
        self.email    = row[1]  # do hàng đã được định sẵn ngay từ lúc put ở trên
        self.name    = row[2]
        self.address = row[3]

# Read the CSV and create User instances
users = [] #tạo mảng users chứa cả file
import csv
with open('user_accounts.csv') as f:
    reader = csv.reader(f)   #đọc file csv lưu tạm vào f
      rows = list(reader)  #rows ở đây lưu lại file thành các hàng và cột tương ứng
theo dạng list
    for row in rows[1:]:    # chạy từng hàng, chạy hết các hàng
        user = User(row) # gọi class để class xử lý dữ liệu ở từng hàng (đã giải thích ở
trên)
        users.append(user)  #append vào users như đã quy ước

# Solution testing
last = users[-1]
print(last.id)
print(last.email)
print(last.name)
print(last.address)
```