AI VIET NAM – COURSE 2024

Python OOP – Exercise

Ngày 23 tháng 6 năm 2024

Ngày thực hiện:	22/06/2024
Người thực hiện:	Đinh Thị Tâm
Nguồn:	AIO2024 - Week3
Nguồn dữ liệu (nếu có):	Link of Data Sources
Từ khóa:	Data structure
Người tóm tắt:	Đinh Thị Tâm

I. Câu hỏi tự luận

1. Câu 1:

   (a) Code

```python
import torch
from torch import nn


class Softmax(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        return torch.exp(x)/torch.sum(torch.exp(x))


class softmax_stable(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        c = torch.max(x)
        return torch.exp(x-c)/torch.sum(torch.exp(x-c))


# main
# Examples 1
data = torch.Tensor([1, 2, 3])
softmax = Softmax()
output = softmax(data)
print(output)
data = torch.Tensor([1, 2, 3])
softmax_stable = softmax_stable()
output = softmax_stable(data)
```

```
print(output)
```

(b) Kết quả thực thi

```
tensor([0.0900, 0.2447, 0.6652])
tensor([0.0900, 0.2447, 0.6652])
```

2. Câu 2

(a) Code

```
from abc import ABC, abstractmethod


class Person(ABC):
    def __init__(self, name, yob):
        self._name = name
        self._yob = yob

    def __str__(self):
        return 'Name: {} - yob: {}'.format(self._name, self._yob)

    @abstractmethod
    def describe(self):
        pass

    def get_yob(self):
        return self._yob


class Doctor(Person):
    def __init__(self, name, yob, specialist):
        super().__init__(name, yob)
        self.__specialist = specialist

    def describe(self):
        print(f'Doctor - {super().__str__()}', end=' ')
        print(f' - specialist: {self.__specialist}')


class Student(Person):
    def __init__(self, name, yob, grade):
        super().__init__(name, yob)
        self.__grade = grade

    def describe(self):
        print(f'Student - {super().__str__()} - grade: {self.__grade}')


class Teacher(Person):
    def __init__(self, name, yob, subject):
        super().__init__(name, yob)
        self.__subject = subject

    def describe(self):
        print(f'Teacher - {super().__str__()} - subject: {self.__subject}')


class Ward:
    def __init__(self, name):
```

```
        self.__lst_person = []
        self.__name = name

    def add_person(self, person):
        self.__lst_person.append(person)

    def describe(self):
        print(f'Ward: {self.__name}, co cac person sau:')
        for x in self.__lst_person:
            x.describe()

    def __call__(self, name):
        self.__name = name

    def count_doctor(self):
        count = 0
        for x in self.__lst_person:
            if type(x) is Doctor:
                count += 1
        return count

    def get_name(self):
        return self.__name
    # sort theo tuoi

    def sort_age(self):
        self.__lst_person.sort(key=lambda x: x._yob)

    def compute_average_teacher(self):
        count_teacher = 0
        sum_age = 0
        for x in self.__lst_person:
            if isinstance(x, Teacher):
                count_teacher += 1
                sum_age += x.get_yob()
        if count_teacher > 0:
            return sum_age/count_teacher
        return None


# main
# (2a)
student1 = Student(name=" studentA ", yob=2010, grade="7")
student1.describe()
teacher1 = Teacher(name=" teacherA ", yob=1969, subject=" Math ")
teacher1.describe()
doctor1 = Doctor(name=" doctorA ", yob=1945, specialist=" Endocrinologists ")
doctor1.describe()
# (2b)
print()
teacher2 = Teacher(name=" teacherB ", yob=1995, subject=" History ")
doctor2 = Doctor(name=" doctorB ", yob=1975, specialist=" Cardiologists ")
# tao Ward
ward = Ward('HCM')
ward.add_person(student1)
ward.add_person(teacher1)
ward.add_person(teacher2)
ward.add_person(doctor1)
ward.add_person(doctor2)
ward.describe()
```

```
# in danh sach cac doctor
print(f'Trong {ward.get_name()} co {ward.count_doctor()} bac si')
print('Danh sach sau khi sap xep:')
ward.sort_age()
ward.describe()
print(f'Trung binh nam sinh cac teacher trong Ward ', end=' ')
print(f'{ward.get_name()} la: {ward.compute_average_teacher()}')
```

(b) Kết quả thực thi

```
     Student - Name:  studentA  - yob: 2010 - grade: 7
Teacher - Name:  teacherA  - yob: 1969 - subject:  Math
Doctor - Name:  doctorA  - yob: 1945  - specialist:  Endocrinologists

Ward: HCM, có các person sau:
Student - Name:  studentA  - yob: 2010 - grade: 7
Teacher - Name:  teacherA  - yob: 1969 - subject:  Math
Teacher - Name:  teacherB  - yob: 1995 - subject:  History
Doctor - Name:  doctorA  - yob: 1945  - specialist:  Endocrinologists
Doctor - Name:  doctorB  - yob: 1975  - specialist:  Cardiologists
Trong phố HCM có 2 bác sĩ
Danh sach sau khi sap xep:
Ward: HCM, có các person sau:
Doctor - Name:  doctorA  - yob: 1945  - specialist:  Endocrinologists
Teacher - Name:  teacherA  - yob: 1969 - subject:  Math
Doctor - Name:  doctorB  - yob: 1975  - specialist:  Cardiologists
Teacher - Name:  teacherB  - yob: 1995 - subject:  History
Student - Name:  studentA  - yob: 2010 - grade: 7
Trung bình năm sinh của teacher trong quận  HCM là: 1982.0
```

3. Câu 3

(a) Code

```
class MyStack:
    def __init__(self, capacity=5):
        self.__stack = []
        self.__capacity = capacity

    def is_empty(self):
        return (len(self.__stack) == 0)

    def is_full(self):
        return (len(self.__stack) == self.__capacity)

    def describe(self):
        print(self.__stack)

    def push(self, x):
        if (self.is_full()):
            return 'Stack is full. Do not thing'
        self.__stack.append(x)

    def pop(self):
        if (self.is_empty()):
            return 'Stack is empty. Do not thing'
```

```
        return self.__stack.pop(-1)

    def top(self):
        if (self.is_empty()):
            return 'Stack is empty. Do not thing'
        return self.__stack[-1]


# test
stack1 = MyStack()
stack1.push(1)
stack1.push(2)
print(stack1.is_full())
print(stack1.top())
print(stack1.pop())
print(stack1.top())
print(stack1.pop())
print(stack1.is_empty())

```

(b) Kết quả thực thi

```
False
2
2
1
1
True
```

4. Câu 4

(a) Code

```
class MyQueue:
    def __init__(self, capacity=5):
        self.__queue = []
        self.__capacity = capacity

    def is_empty(self):
        return (len(self.__queue) == 0)

    def is_full(self):
        return (len(self.__queue) == self.__capacity)

    def describe(self):
        print(self.__queue)

    def enqueue(self, x):
        if (self.is_full()):
            return 'Queue is full. Do not thing'
        self.__queue.append(x)

    def dequeue(self):
        if (self.is_empty()):
            return 'Queue is empty. Do not thing'
        return self.__queue.pop(0)

    def front(self):
        if (self.is_empty()):
```

```
            return 'Queue is empty. Do not thing'
        return self.__queue[0]


# test
queue1 = MyQueue(capacity=5)

queue1.enqueue(1)
queue1.enqueue(2)
print(queue1.is_full())
print(queue1.front())
print(queue1.dequeue())
print(queue1.front())
print(queue1.dequeue())
print(queue1.is_empty())

```

(b) Kết quả thực thi

```
False
1
1
2
2
True
```

II. Câu hỏi trắc nghiệm

1. Câu 1:

```
    import torch
import torch .nn as nn
data = torch . Tensor ([1 , 2, 3])
softmax_function = nn. Softmax (dim =0)
output = softmax_function ( data )
assert round ( output [0]. item () , 2) == 0.09
print(output)
```

2. Câu 2:

```
      import torch
import torch .nn as nn


class MySoftmax (nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        # Your Code Here
        return torch.exp(x)/torch.sum(torch.exp(x))

    # End Code Here
data = torch.Tensor([5, 2, 4])
my_softmax = MySoftmax()
output = my_softmax(data)
assert round(output[-1]. item(), 2) == 0.26
print(output)
```

3. Câu 3:

```
      import torch
import torch .nn as nn


class MySoftmax (nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, x):
        # Your Code Here
        return torch.exp(x)/torch.sum(torch.exp(x))

    # End Code Here
data = torch.Tensor ([1 , 2, 300000000])
my_softmax = MySoftmax()
output = my_softmax(data)
assert round ( output [0]. item () , 2) == 0.0
print(output)
```

4. Câu 4:

```
    import torch
import torch.nn as nn


class SoftmaxStable (nn.Module):
    def __init__(self):
        super(). __init__()

    def forward(self, x):
        x_max = torch .max(x, dim=0, keepdims=True)
        x_exp = torch .exp(x - x_max . values)
        partition = x_exp .sum(0, keepdims=True)
        return x_exp / partition


data = torch.Tensor([1, 2, 3])
softmax_stable = SoftmaxStable()
output = softmax_stable(data)
assert round(output[-1]. item(), 2) == 0.67
print(output)

```

5. Câu 5:

```
      from abc import ABC, abstractmethod


class Person(ABC):
    def __init__(self, name, yob):
        self._name = name
        self._yob = yob

    def __str__(self):
        return 'Name: {}, yob: {}'.format(self._name, self._yob)

    @abstractmethod
    def describe(self):
        pass
```

```
15
16      def get_yob(self):
17          return self._yob
18
19
20  class Student(Person):
21      def __init__(self, name, yob, grade):
22          super().__init__(name, yob)
23          self.__grade = grade
24
25      def describe(self):
26          print(f'Student - {super().__str__()}, grade: {self.__grade}')
27
28
29  # test
30  student1 = Student(name=" studentZ2023 ", yob=2011, grade="6")
31  assert student1._yob == 2011
32  student1.describe()
```

6. Câu 6:

```
1   from abc import ABC, abstractmethod
2
3
4   class Person(ABC):
5       def __init__(self, name, yob):
6           self._name = name
7           self._yob = yob
8
9       def __str__(self):
10          return 'Name: {}, yob: {}'.format(self._name, self._yob)
11
12      @abstractmethod
13      def describe(self):
14          pass
15
16      def get_yob(self):
17          return self._yob
18
19
20  class Teacher(Person):
21      def __init__(self, name, yob, subject):
22          super().__init__(name, yob)
23          self.__subject = subject
24
25      def describe(self):
26          print(f'Teacher - {super().__str__()}, subject: {self.__subject}')
27
28
29  teacher1 = Teacher(name=" teacherZ2023 ", yob=1991, subject=" History ")
30  assert teacher1 . _yob == 1991
31  teacher1 . describe()
```

7. Câu 7:

```
1       from abc import ABC, abstractmethod
2
3
4   class Person(ABC):
5       def __init__(self, name, yob):
```

```
        self._name = name
        self._yob = yob

    def __str__(self):
        return 'Name: {}, yob: {}'.format(self._name, self._yob)

    @abstractmethod
    def describe(self):
        pass

    def get_yob(self):
        return self._yob


class Doctor(Person):
    def __init__(self, name, yob, specialist):
        super().__init__(name, yob)
        self.__specialist = specialist

    def describe(self):
        print(f'Doctor - {super().__str__()}, specialist: {self.__specialist}')

doctor1 = Doctor ( name =" doctorZ2023 ", yob =1981 , specialist ="
    Endocrinologists ")
assert doctor1 . _yob == 1981
doctor1.describe ()
```

8. Câu 8:

```
from abc import ABC, abstractmethod


class Person(ABC):
    def __init__(self, name, yob):
        self._name = name
        self._yob = yob

    def __str__(self):
        return 'Name: {}, yob: {}'.format(self._name, self._yob)

    @abstractmethod
    def describe(self):
        pass

    def get_yob(self):
        return self._yob


class Doctor(Person):
    def __init__(self, name, yob, specialist):
        super().__init__(name, yob)
        self.__specialist = specialist

    def describe(self):
        print(f'Doctor - {super().__str__()}, specialist: {self.__specialist}')


class Student(Person):
    def __init__(self, name, yob, grade):
        super().__init__(name, yob)
        self.__grade = grade
```

```
33
34      def describe(self):
35          print(f'Student - {super().__str__()}, grade: {self.__grade}')
36
37
38  class Teacher(Person):
39      def __init__(self, name, yob, subject):
40          super().__init__(name, yob)
41          self.__subject = subject
42
43      def describe(self):
44          print(f'Teacher - {super().__str__()}, subject: {self.__subject}')
45
46
47  class Ward:
48      def __init__(self, name):
49          self.__lst_person = []
50          self.__name = name
51
52      def add_person(self, person: Person):
53          self.__lst_person.append(person)
54
55      def describe(self):
56          print(f'Ward: {self.__name}, co cac person sau:')
57          for x in self.__lst_person:
58              x.describe()
59
60      def __call__(self, name):
61          self.__name = name
62
63      def count_doctor(self):
64          count = 0
65          for x in self.__lst_person:
66              if type(x) is Doctor:
67                  count += 1
68          return count
69
70      def get_name(self):
71          return self.__name
72      # sort theo tuoi
73
74      def sort_age(self):
75          self.__lst_person.sort(key=lambda x: x._yob)
76
77      def compute_average(self):
78          sum_age = sum(item._yob for item in self.__lst_person)
79          return sum_age/len(self.__lst_person)
80
81
82  # main
83  student1 = Student(name=" studentA ", yob=2010, grade="7")
84  teacher1 = Teacher(name=" teacherA ", yob=1969, subject=" Math ")
85  teacher2 = Teacher(name=" teacherB ", yob=1995, subject=" History ")
86  doctor1 = Doctor(name=" doctorA ", yob=1945, specialist=" Endocrinologists ")
87  doctor2 = Doctor(name=" doctorB ", yob=1975, specialist=" Cardiologists ")
88  ward1 = Ward(name=" Ward1 ")
89  ward1.add_person(student1)
90  ward1.add_person(teacher1)
91  ward1.add_person(teacher2)
92  ward1.add_person(doctor1)
```

```
assert ward1 . count_doctor() == 1
ward1.add_person(doctor2)
print(ward1 . count_doctor())
```

9. Câu 9:

```
class MyStack:
    def __init__(self, capacity=5):
        self.__stack = []
        self.__capacity = capacity

    def is_full(self):
        return (len(self.__stack) == self.__capacity)

    def push(self, value):
        if (self.is_full()):
            return 'Stack is full. Do not thing'
        self.__stack.append(value)


# test
stack1 = MyStack(capacity=5)
stack1 . push(1)
assert stack1 . is_full() == False
stack1 . push(2)
print(stack1 . is_full())

```

10. Câu 10:

```
class MyStack:
    def __init__(self, capacity=5):
        self.__stack = []
        self.__capacity = capacity

    def is_full(self):
        return (len(self.__stack) == self.__capacity)

    def push(self, value):
        if (self.is_full()):
            return 'Stack is full. Do not thing'
        self.__stack.append(value)

    def top(self):
        if (len(self.__stack) == 0):
            return 'Stack is empty. Do not thing'
        return self.__stack[-1]


# test
stack1 = MyStack(capacity=5)
stack1 . push(1)
assert stack1 . is_full() == False
stack1 . push(2)
print(stack1.top())

```

11. Câu 11:

```
class MyQueue:
```

```
    def __init__(self, capacity=5):
        self.__queue = []
        self.__capacity = capacity

    def is_full(self):
        return (len(self.__queue) == self.__capacity)

    def enqueue(self, x):
        if (self.is_full()):
            return 'Queue is full. Do not thing'
        self.__queue.append(x)

queue1 = MyQueue ( capacity =5)
queue1 . enqueue (1)
assert queue1 . is_full () == False
queue1 . enqueue (2)
print ( queue1 . is_full ())
```

12. Câu 12:

```
class MyQueue:
    def __init__(self, capacity=5):
        self.__queue = []
        self.__capacity = capacity

    def is_empty(self):
        return (len(self.__queue) == 0)

    def is_full(self):
        return (len(self.__queue) == self.__capacity)

    def dequeue(self):
        if (self.is_empty()):
            return 'Queue is empty. Do not thing'
        return self.__data.pop(0)

    def enqueue(self, x):
        if (self.is_full()):
            return 'Queue is full. Do not thing'
        self.__queue.append(x)

    def front(self):
        if (self.is_empty()):
            return 'Queue is empty. Do not thing'
        return self.__queue[0]


queue1 = MyQueue(capacity=5)
queue1 . enqueue(1)
assert queue1 . is_full() == False
queue1 . enqueue(2)
print(queue1.front())

```