ECE 326 Tutorial 5

Python Review

Prepared by: Wendy Qiu
October 13, 2020

Outline

1. Midterm Questions Review
2. Exercise 4 Questions Review

Midterm One: Q1

```
>> mystery[1::3]
'ruum'
>> mystery[-1::-2]
'mlcuec'
>> mystery[3:6]
'pus'
>> mystery.count('u')
3
```

➔ What is the mystery string?

?	**r**	?	?	**u**	?	?	**u**	?	?	**m**
c	r	**e**	?	**u**	?	**c**	u	**l**	?	**m**
c	r	e	**p**	**u**	**s**	c	u	l	?	m
c	r	e	p	**u**	s	c	**u**	l	**u**	m

Midterm One: Q2

```
>> result = unpack(fmt,b'\x00\x00\x00\x01john\x00\x00\x00\x00' \ ..
b'\x00\x00\x00\x02\x00\x00\x00\xff’)
```

```
>> print(result) (1, b'john', 2, 255)
```

➔ What is the format string, fmt, that will generate the following output?

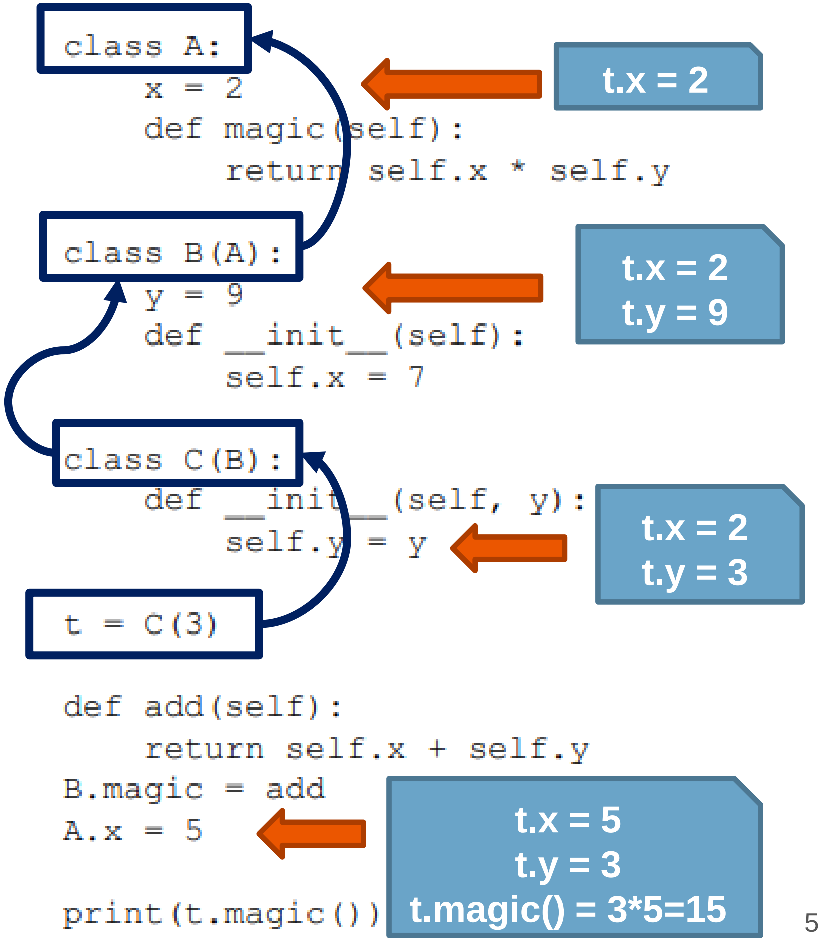
- Same order ➔ big endian: ***!*** Or **>**
- One integer (4 bytes): ***i***
- String/Chars (length = 4 bytes): ***4s***
- Long long (8 bytes): ***q***
- One integer (4 bytes): ***i***

Answer: ‘!i4sqi’ or ‘>i4sqi’, or ‘>xxxb4sqi’ (using padding)

(Example M1Q3)

Midterm One: Q3

Question: If we assume that Python uses *dynamic dispatch* to implement runtime polymorphism, what would be the output of this program?



```
class A:
  x = 2
  def magic(self):
    return self.x * self.y
class B(A):
  y = 9
  def __init__(self):
    self.x = 7
class C(B):
  def __init__(self, y):
    self.y = y
t = C(3)

def add(self):
  return self.x + self.y
B.magic = add
A.x = 5

print(t.magic())
```

Midterm One: Q4 (Example M1Q4)

Question: Complete the class definition for the Point class below by adding more method(s), such that the above output will be printed.

Given the follow expected output:

```
>> p1 = Point(2,3)
>> p2 = Point(3,4)
>> print(p1 + p2)
(5, 7)
```

```
class Point:
    def __init__(self, x=0, y=0):
        self.x, self.y = x, y

    # complete me here
```

Midterm One: Q5 - a

Apply the three criteria of first class citizenship to classes in C++:

1. Can be used? ✗
    1. Cannot be assigned to a variable
    2. Cannot be passed to or return from a function
2. Can be constructed?
    1. Yes, you can create classes in local scope (i.e., inside a function)
    2. No, you cannot create new classes at runtime
3. Have a type? ➔ No, class itself does not have a type ✗

Midterm One: Q5 - b

Based on the analysis above, are classes in C++ first class citizens? Why or why not?

- No
- Need to satisfy all three criteria in order to be a first class citizen

➔ Class in C++ does not pass all three criteria

Midterm One: Q6 - a

```
hour = Bound('hour', 0, 24)
minute = Bound('minute', 0, 60)
second = Bound('second', 0, 60)
```

Complete the Bound descriptor class such that the value set to the descriptor must be within the range *[min, max).*

Raise ValueError if the assigned value is out of bound. Note that you do not need to implement the *__delete__* method.

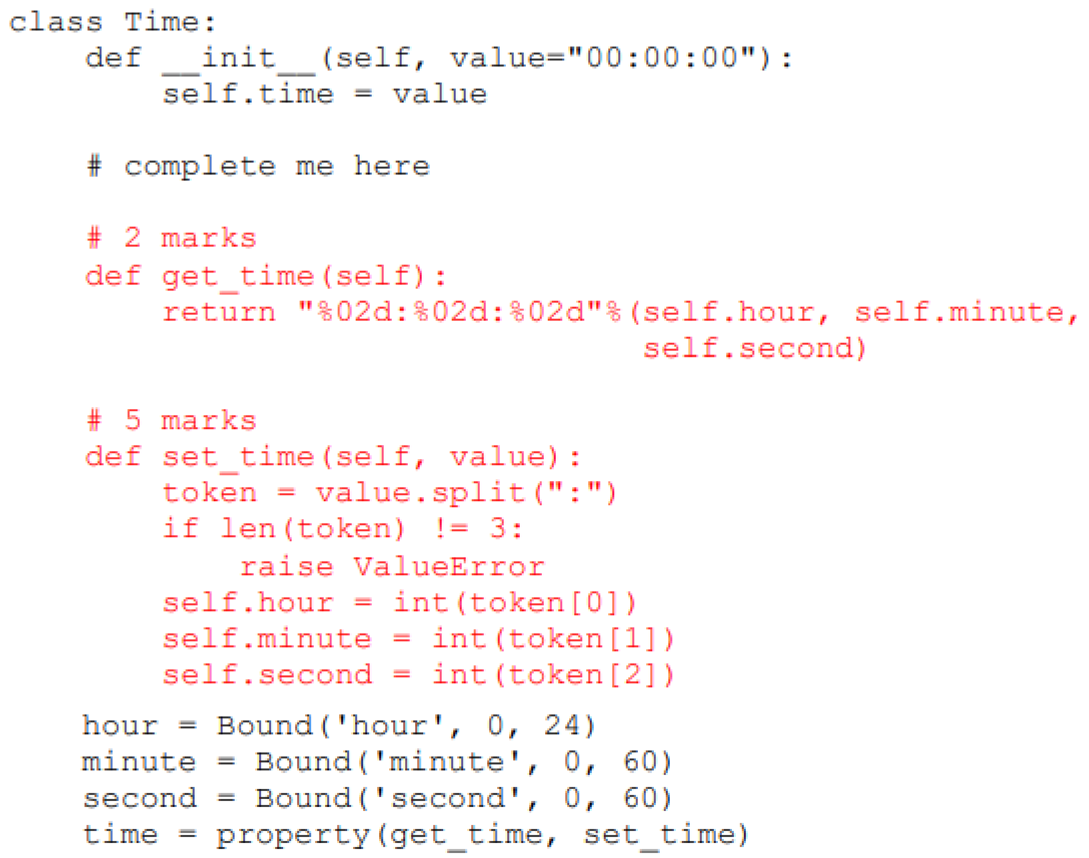
```
class Bound:
    def __init__(self, name, min, max):
        self.name = "_" + name
        self.min = min
        self.max = max

    # complete me here

    # 3 marks
    def __get__(self, inst, owner):
        return getattr(inst, self.name)

    # 4 marks
    def __set__(self, inst, value):
        if value < self.min or value >= self.max:
            raise ValueError
        setattr(inst, self.name, value)
```

Q6 - b

Complete the set_time and get_time methods of Time class.

Hint: Recall that int() will raise a ValueError if a string cannot be converted to integer

Sample Outputs:
>> t = Time("07:02:30")
>> t.hour, t.minute, t.second
(7, 2, 30)
>> t.time '07:02:30’
>> t.time = "hello world"
ValueError



```
class Time:
    def __init__(self, value="00:00:00"):
        self.time = value

    # complete me here

    # 2 marks
    def get_time(self):
        return "%02d:%02d:%02d"%(self.hour, self.minute,
                                 self.second)

    # 5 marks
    def set_time(self, value):
        token = value.split(":")
        if len(token) != 3:
            raise ValueError
        self.hour = int(token[0])
        self.minute = int(token[1])
        self.second = int(token[2])
    hour = Bound('hour', 0, 24)
    minute = Bound('minute', 0, 60)
    second = Bound('second', 0, 60)
    time = property(get_time, set_time)
```

Outline

1. Midterm Questions Review

2. Exercise 4 Questions Review

Exercise 4 - Q1:

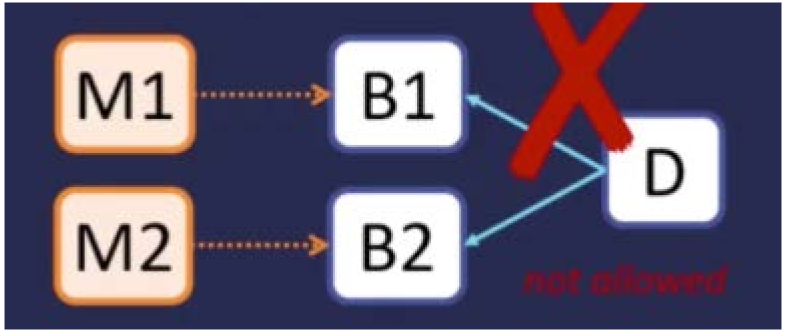
1. `type` is to classes as `object` is to instances. [✓ T] F

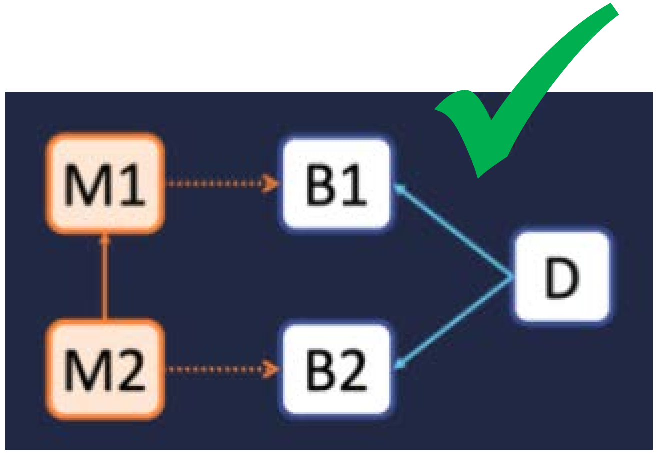
2. In multiple inheritance, TypeError is raised when there is a shared base metaclass. T F

3. `vars(self)` returns `self.__dict__`. T F

Exercise 4 - Q1 - 2:

2. In multiple inheritance, TypeError is raised when there is a shared base metaclass. T [✓ F]





1. `type` is to classes as `object` is to instances. [✓ T] F

2. In multiple inheritance, TypeError is raised when there is a shared base metaclass. T [✓ F]

3. `vars(self)` returns `self.__dict__`. [✓ T] F

4. The `__delete__` special method is also known as the destructor. T [✓ F]

5. A metaclass does not need to be a class; it can also be a function. [✓ T] F

Exercise 4 – Q2a:

a) Which of the following statements about descriptors or properties are true?

i. A descriptor can manage multiple attributes at once, a property can only manage one. ➔ **False**

ii. A descriptor with only __get__ can be overwritten or deleted, a property with only getter cannot. ➔ **True**

Exercise 4 – Q2a:

iii. A descriptor can manage a method, a property cannot (data attribute only). → **False**

iv. A descriptor keeps data within its own instance, a property uses that of the parent instance. → **False**

v. A descriptor with only __set__ has the same behaviour as a property with only setter. → **False** (Example E4Q2_5)

Exercise 4 – Q2b:

b) Which of the following about Python metaclass are true?

i. The __call__ method of a metaclass initiates the process of creating a new class. ➔ **False**

ii. The __new__ method of a metaclass instantiates new objects for its classes. ➔ **False**

Exercise 4 – Q2a:

iii. To avoid infinite recursion, a metaclass cannot have its own metaclass. → **False**

iv. Like regular classes, multiple inheritance is supported for metaclasses. → **False**

v. During name resolution, a class's metaclass is looked up before its super classes are searched. → **False**

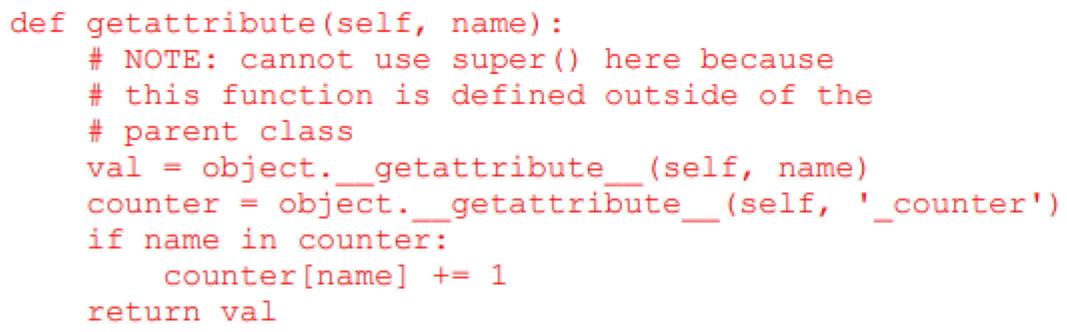
Exercise 4 – Q3:

Differences between the following built-in methods:

- *__set__:* used by a descriptor to manage the assignment of one attribute on an instance of a different class.
- *__setitem__:* used when overloading assignment to an index
- *__setattr__:* used to manage attribute assignment to instances of a class

Exercise 4 – Q4 - a:

Write a metaclass that counts how many times a method is called.



```
def getattribute(self, name):
    # NOTE: cannot use super() here because
    # this function is defined outside of the
    # parent class
    val = object.__getattribute__(self, name)
    counter = object.__getattribute__(self, '_counter')
    if name in counter:
        counter[name] += 1
    return val
```

- Defined outside of the metaclass because, if defined inside, when a class attribute is used, will call __new__() immediately without knowing the implementation of getattribute(), so the attrs['__getattribute__'] cannot be overloaded

- Enters each time an attribute of the target class instance gets called
- Checks if there is an existing counter for this attribute: increment the counter by 1

Exercise 4 – Q4 - a:

Write a metaclass that counts how many times a method is called.

```
class MethodCounter(type):
    def __new__(mcs, name, base, attrs):
        assert('_counter' not in attrs)
        counter = {}
        for a_name in attrs:
            if callable(attrs[a_name]):
                counter[a_name] = 0
        attrs['_counter'] = counter
        attrs['__getattribute__'] = getattribute
        return super().__new__(mcs, name, base, attrs)

    def get_count(cls, name):
        return cls._counter.get(name, 0)
```

Check if already called __new__()
Create empty dict for counter if not

Set each individual counter to 0

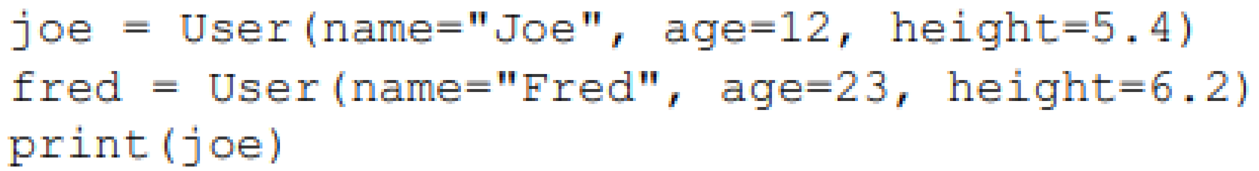
Assign counter dict to the according class

Overload __getattribute__()

Create a new metaclass

Find the value of the counter for a specific method using the name

Exercise 4 – Q4 - b: (Example E4Q4b)



```
joe = User(name="Joe", age=12, height=5.4)
fred = User(name="Fred", age=23, height=6.2)
print(joe)
```

1) **Output:** Fred: Age 23, Height 6.200000 ➔ *self.value = value*

2) **Output:** Joe: Age Joe, Height Joe, Fred: Age Fred, Height Fred

   ➔ *inst.value = value*

3) **Maximum recursion depth exceeded** ➔ change *name* to *_name*

Thanks for listening!