ECE 326 Tutorial 6

Exercise 5 Review

Prepared by: Wendy Qiu
October 21, 2020

Exercise 5 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

1. Method resolution order helps Python deal with the repeated base class problem.

T **F**

- No repeated base class problem in Python!

Exercise 5 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

2. All Python built-in methods go through type slots.

T **F**

Not all built-in methods go through type slots!
E.g. __prepare__

Exercise 5 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

3. A programming language without multiple inheritance cannot implement mixins.

T **F**

Mixin: Code reuse without becoming the parent class
Inclusion rather than inheritance!

Exercise 5 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

4. Python does not have class scope.

T **F**

This is problematic if multiple classes in inheritance use same name for different purposes! ➔ Mixins

Exercise 5 - Q1:

Question 1. True or False

Circle **T** if the statement is true, otherwise circle **F** if the statement is false.

5. You can mix the use of fully qualified names and super() in cooperative inheritance.

T **F**

Two ways to initialize **ALL** immediate super classes:
1. Fully qualified name
2. super()

Exercise 5 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following are true about Python mixins?

1. It requires subclass to complete its implementation.

➔ **False**

Exercise 5 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following are true about Python mixins?

2. It can contain both member variables and functions.

➔ **True**

Exercise 5 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following are true about Python mixins?

3. It is used as a super type to the derived class.

➔ **True**

Exercise 5 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following are true about Python mixins?

4. Using it requires method forwarding.

➔ **False**

Exercise 5 – Q2a:

Question 2. Multiple Choices

Pick all answer(s) that are correct.

a) Which of the following are true about Python mixins?

5. The order in which mixins are inherited may change behaviour of the subclass.

➔ **True**

Exercise 5 – Q2b:

Question 2. Multiple Choices

b) Which of the following statements about method resolution order (MRO) are true?

1. Object-oriented programming languages that perform late binding must implement MRO. ➔ **True**

Exercise 5 – Q2b:

Question 2. Multiple Choices

b) Which of the following statements about method resolution order (MRO) are true?

2. Depth-first search, left to right, respects local precedence order. ➔ **True**

Exercise 5 – Q2b:

Question 2. Multiple Choices

b) Which of the following statements about method resolution order (MRO) are true?

3. In Python, method resolution order is also used on data attributes. ➔ **True**

Exercise 5 – Q2b:

Question 2. Multiple Choices

b) Which of the following statements about method resolution order (MRO) are true?

4. C3 linearization cannot fail while creating method resolution order. ➔ **False**

Exercise 5 – Q2b:

Question 2. Multiple Choices

b) Which of the following statements about method resolution order (MRO) are true?

5. Linearization is the result of applying method resolution order to inheritance hierarchy. ➔ **False**

Exercise 5 – Q3:

a) List two properties that C3 Linearization guarantees. You must explain what each guarantee means with an example.

- **Local precedence order:** the order in which the parent classes are inherited
- **Preserves monotonicity:** linearization of a class will not change order regardless of the inheritance hierarchy it may be in

```
class Person:
    def vacation(self):
        return "Toronto Islands"

class Student(Person):
    def vacation(self):
        return "Queen's Park"

class PartTime(Person, Student):
    pass
```

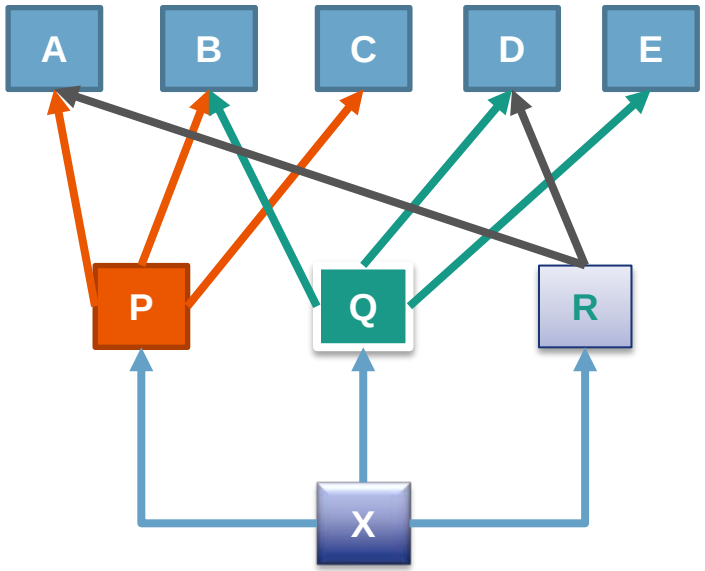
Under new algorithm, the MRO is:
(PartTime, Student, Person)

```
>> PartTime(…).vacation()
Queen's Park
```

Q: But Student is more specialized than Person?
A: Yes, as such, this inheritance hierarchy is *ambiguous.*

b) C3 Linearization:

```
class A: pass
class B: pass
class C: pass
class D: pass
class E: pass
class P(A, B, C): pass
class Q(D, B, E): pass
class R(D, A): pass
class X(P, R, Q): pass
```



b) C3 Linearization:

```
class A: pass
class B: pass
class C: pass
class D: pass
class E: pass
class P(A, B, C): pass
class Q(D, B, E): pass
class R(D, A): pass
class X(P, R, Q): pass
```

```
L[A] = (A, o)                L[B] = (B, o)
L[C] = (C, o)                L[D] = (D, o)
L[E] = (E, o)                L[P] = (P, A, B, C, o)
L[Q] = (Q, D, B, E, o)       L[R] = (R, D, A, o)
```

➔ L[X]

```
= (X, merge((P, A, B, C, o), (R, D, A, o), (Q, D, B, E, o), (P, R, Q)))
= (X, P, merge((A, B, C, o), (R, D, A, o), (Q, D, B, E, o), (R, Q)))
= (X, P, R, merge((A, B, C, o), (D, A, o), (Q, D, B, E, o), (Q)))
# A bad head, in tail of 2nd list
= (X, P, R, Q, merge((A, B, C, o), (D, A, o), (D, B, E, o)))
= (X, P, R, Q, D, merge((A, B, C, o), (A, o), (B, E, o)))
= (X, P, R, Q, D, A, merge((B, C, o), (o), (B, E, o)))
= (X, P, R, Q, D, A, B, merge((C, o), (o), (E, o)))
= (X, P, R, Q, D, A, B, C, E, o)
```

c) C3 Linearization:

```
class A : foo = 1          class N(type) : foo = 11
class B : qux = 2          class M(N) : qux = 7
class W(A) : baz = 3       class P(W, X, Y) : pass
class X(A) : bar = 4       class Q(X, Y, Z): bar = 9
class Y(B) : foo = 5       class R(W, Z) : bar = 0
class Z(B) : baz = 6       class H(P, Q, R, metaclass=M) : pass
```

L[A] = (A, o) L[B] = (B, o) L[W] = (W, A, o)
L[X] = (X, A, o) L[Y] = (Y, B, o) L[Z] = (Z, B, o)
L[P] = (P, merge((W, A, o), (X, A, o), (Y, B, o), (W, X, Y)))
L[P] = (P, W, X, merge((A, o), (A, o), (Y, B, o), (Y)))
L[P] = (P, W, X, A, merge((o), (o), (Y, B, o), (Y)))
L[P] = (P, W, X, A, Y, B, o)
L[Q] = (Q, merge((X, A, o), (Y, B, o), (Z, B, o), (X, Y, Z)))
L[Q] = (Q, X, A, merge((o), (Y, B, o), (Z, B, o), (Y, Z)))
L[Q] = (Q, X, A, Y, Z, B, o)
L[R] = (R, merge((W, A, o), (Z, B, o), (W, Z))) = (R, W, A, Z, B, o)

c) C3 Linearization:

```
class H(P, Q, R, metaclass=M) : pass
```

L[P] = (P, W, X, A, Y, B, o)
L[Q] = (Q, X, A, Y, Z, B, o)
L[R] = (R, W, A, Z, B, o)

➔
L[H] = (H, merge((P, W, X, A, Y, B, o), (Q, X, A, Y, Z, B, o), (R, W, A, Z, B, o), (P, Q, R)))
L[H] = (H, P, Q, R, merge((W, X, A, Y, B, o), (X, A, Y, Z, B, o), (W, A, Z, B, o)))
L[H] = (H, P, Q, R, W, X, merge((A, Y, B, o), (A, Y, Z, B, o), (A, Z, B, o)))
L[H] = (H, P, Q, R, W, X, A, Y, merge((B, o), (Z, B, o), (Z, B, o)))
L[H] = (H, P, Q, R, W, X, A, Y, Z, B, o)

c) C3 Linearization:

```
class A : foo = 1          class N(type) : foo = 11
class B : qux = 2          class M(N) : qux = 7
class W(A) : baz = 3       class P(W, X, Y) : pass
class X(A) : bar = 4       class Q(X, Y, Z): bar = 9
class Y(B) : foo = 5       class R(W, Z) : bar = 0
class Z(B) : baz = 6       class H(P, Q, R, metaclass=M) : pass
```

Q: What are the values of H.foo, H.bar, H.baz, and H.qux? (Code Verification)

From previous: L[H] = (H, P, Q, R, W, X, A, Y, Z, B, o)

foo: A, B, N ⇒ A ⇒ foo = **1**
bar: X, Q, R ⇒ Q ⇒ bar = **9**
baz: W, Z ⇒ W ⇒ baz = **3**
qux: B, M ⇒ B ⇒ qux = **2**

Exercise 5 – Q4 - a:

a) Write a SocketLogMixin that will send log messages to a server, and a FileLogMixin that will save log messages in a local file.

```
class FileLogMixin(LogMixin):
    def __init__(self, log_file, **kwargs):
        self.file = open(log_file, "wt")
        super().__init__(**kwargs)

    def _write(self, msg):
        self.file.write(msg + '\n')

    def __del__(self):
        self.file.close()
```

Exercise 5 – Q4 - a:

```
class SocketLogMixin(LogMixin):
    def __init__(self, log_server, **kwargs):
        host, port = log_server.split(":")
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((host, int(port)))
        super().__init__(**kwargs)

    def _write(self, msg):
        self.sock.send(msg + '\n')

    def __del__(self):
        self.sock.close()
```

Exercise 5 – Q4 - b:

b) Create a mixin named Indexable that will automatically forward __setitem__ and __getitem__ to an instance variable named data.

```
class Indexable:
    def __setitem__(self, index, value):
        self.data[index] = value

    def __getitem__(self, index):
        return self.data[index]
```

Exercise 5 – Q4 - c: (Code Example)

c) Implement a class, CalendarClock, that is derived from Calendar and Clock shown below, and implement three methods, __init__, __str__, and tick().

```
class Clock:
  def __init__(self, h, m, s)
  def __str__(self)

  # moves time forward by 1 second
  # returns True if clock resets,
  # i.e. from 23:59:59 to 00:00:00,
  # False otherwise.
  def tick()
```

```
class Calendar:
  def __init__(self, y, m, d)
  def __str__(self)


  # moves time forward by 1 day
  def advance()
```

Thanks for listening!