EXERCISE 7

Diagramming Conventions

All of our examples below use the following example

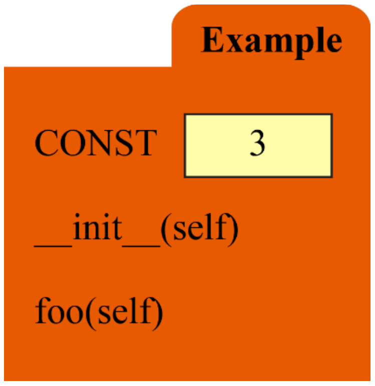
```
   class Example(object):
     CONST=3
     def __init__(self):
1.        pass
     def foo(self):
1.        return 42
```

The red numbers are line numbers for the purposes of method call frames. To make things easier (and more familiar to you), we always assume the first line of a method is line number 1.

Diagramming Class Folders

When you define a class, Python creates a special folder called a **class folder** that stores the information about that class. Technically, Python also creates a global variable with the same name as the class and puts the name of that folder in the global variable; that is what you see when you put the class above in the Python visualizer.

To simplify things, we do not want you to show the variable in global space (just like we never ask you to show the variable for a function name in global space). We just want you create a folder in heap space, and put the name of the class in the tab. We also want you to put the tab on the right so that we can tell this is a class folder. This is illustrated below.



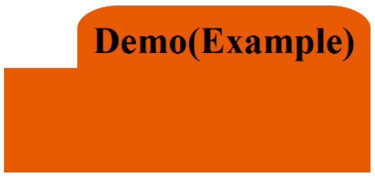
The class folder contains all the methods and any class attributes. In a class definition, any assignment statement outside of a method definition creates a class attribute. In the example above, CONST is a class attribute.

Technically, the methods are also variables themselves. This is what you would see if you used the Python visualizer. However, the contents of this variable are really complicated, and the Python visualizer just writes the method header (the name plus **all** of its parameters) as its value. Therefore, that is what we want you to write for the methods in a class: the method header.

The example above works when the class is a *base case* (e.g. a subclass of class object). Suppose we have a class that is a subclass of a user-defined class, like the example Demo below.

```
class Demo(Example):
  pass
```

We want to indicate this relationship in our class folders. To do this, we put the name of the parent class in parentheses, as shown below. Note that this class has no methods or class attributes in it, because there are none in the class definition.

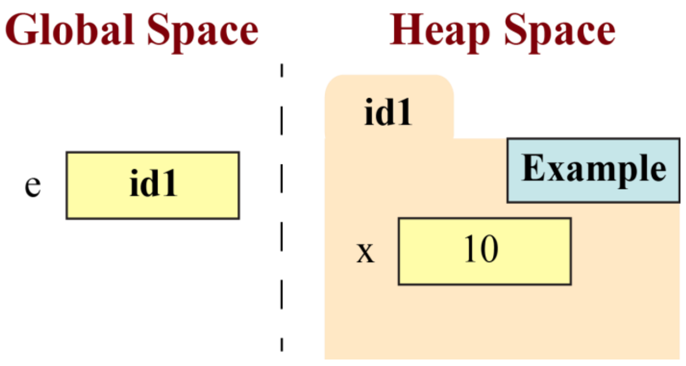


Diagramming Object Folders

A object is *instance* of a class. While there is only one class **Example**, there may be more than one Example object. Each time you call the constructor, you create a new object. For example, suppose we execute the following statements:

```
e = Example()
e.x = 10
```

The constructor for Example creates an empty folder (why?), while the second assignment statement adds an attribute x to the folder. The end result is as follows:



The instance folder only contains attributes added to the instance. It does not contain class attributes, and (for this course) it never contains methods.

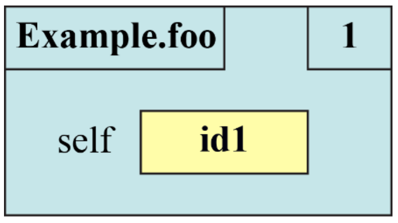
Diagramming Method Calls

A method class works **almost exactly** like a function call. You create the call frame, and plug the arguments into the parameters. You then execute each line in the method, changing the call frame as necessary. Finally, you erase the frame.

The only real difference is the function name that goes in the upper left box. We want both the name of the class *and* the method name in this box. For example. for the method call

```
  e.foo()
```

the initial diagram for the call frame is as follows



Note the name is not foo in the top left folder, but Example.foo. That tells us to use the version of foo that is stored in the class folder Example.

Diagramming Constructor Calls

A constructor is just a function, so Python should create a call frame when it is called. However, this function is built-in, so we do not really know what it looks like. You will notice that the Python visualizer does not create a call frame for the constructor either. However, it does create a call frame for the initializer __init__, which the constructor uses as a helper.

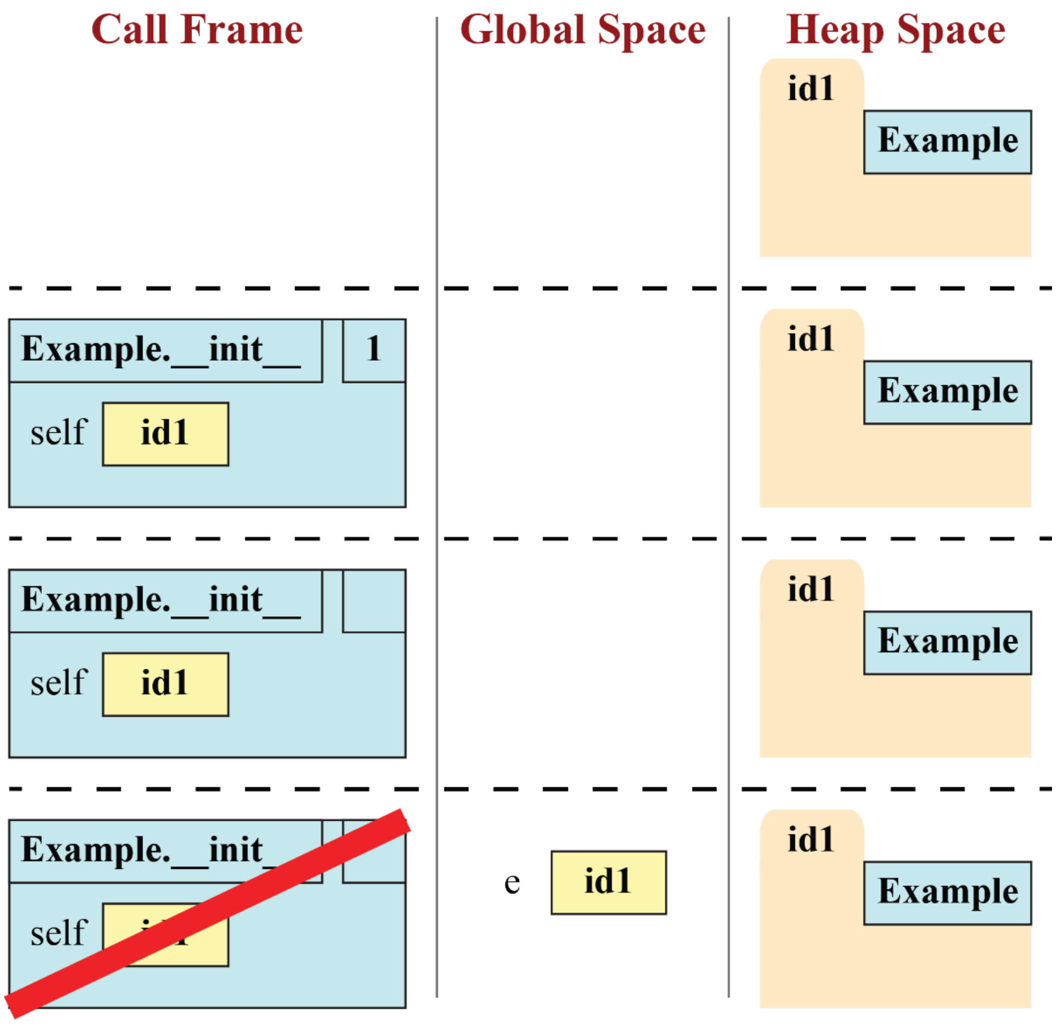
We are going to do it the way that the Python visualizer does it. Remember the 4 steps of a constructor call:

1. It creates a new empty folder of that class.
2. It puts the folder into heap space
3. It executes the method __init__ (creating a call frame)
4. It returns the folder name

In our diagram, we combine steps 1 and 2 (because otherwise we do not know where to put the folder). We also combine step 4 with the erasing step of __init__ **even though __init__ has no return variable**. For example, if we executed the constructor

```
e = Example()
```

The resulting diagram would be as follows.



The first diagram is the folder creation step, before the method __init__ is called. The next two diagrams are for the call frame for __init__. The first picture is at the start of the method, and the second picture is the execution of the one an only line of the method (which does nothing). In the final diagram, we erase the frame and assign the value to our variable.

Assignment Instructions

This entire assignment will involve the following two mysterious (and undocumented) classes.

```
   class A(object):
      x=3
      y=5
      def __init__(self,y):
1.        self.y = y
      def f(self):
1.        return self.y+self.y
      def g(self):
1.        return self.x+self.y
```

```
   class B(A):
      y=4
      z=10
      def __init__(self,x,y):
1.        self.x = x
2.        super().__init__(y+1)
      def f(self):
1.        a = self.g()
2.        return self.z+a
```

The red numbers are line numbers for the purposes of method call frames. To make things easier (and more familiar to you), we always assume the first line of a method is line number 1. We have kept all of these methods short to cut down on the amount that you have to draw.

Part A: Diagram the two Class Folders

The instructions are in them title for this section. We just want you to draw two class folder. Once for the class A and another for the class B. Do not draw global space (even though there are global variables A and B that refer to these class folders). There is also no call frame to draw.

However, the folder for class B requires the subclasses lecture.

Part B: Diagram the Execution of p = A(1)

Draw the execution of the statement p = A(1). This is a constructor call, so it will involve the creation of a call frame for __init__. As usual, you will need a diagram for when the frame for __init__ created, and another when it is erased. In between, you will diagram the lone line of code that is executed.

However, you do not draw the call frame right away. You need a step creating the folder first. In particular, you will need to draw how global space and heap space changes with the call frame, just like you did on the prelim. You do not need to add anything to global space beyond what is required for the statement p = A(1). In particular, you do not need a global variable for class A.

You can complete this part before the Tuesday, November 1 lecture.

Part C: Diagram the Execution of q = B(7,3)

This problem is exactly like Part B. We need the evolution of the global space, heap space, and the call frame for __init__. The difference this time is that there is a call stack created with more than one frame (just like on the first prelim). You should draw that properly.

You do **not** need to include p (from Part B) in your diagram of global space.

Part D: Diagram the Execution of r = q.f()

In this problem, q is the variable from Part C. Diagram the evolution of the call frame for method f. This is a normal method call (not a constructor). If this method calls another method as a helper, you should draw the complete call stack, just like you did in the previous problem.

Your diagrams should include the contents of global space at each step. You do not need to draw heap space, since method f will not change the contents of any folder.