DIAGRAMING CALL FRAMES

SOLUTION FOR ASSIGNMENT A2

Part A

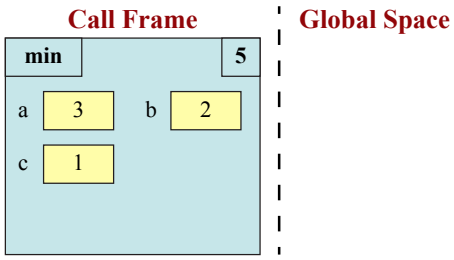
When we execute the call, the parameter `a` gets the value 3, `b` gets the value 2, and `c` gets the value 1. Hence the conditional in line 6 is true, so line 7 is executed. So our diagram sequence consists of six steps:

- The diagram of the call frame when it starts.
- The intermediate diagrams for the call frame as we execute each line 5-7 and line 10.
- A final diagram crossing out the call frame, indicating its deletion.

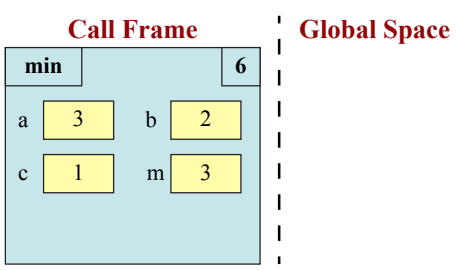
Note that the contents of this call frame are all `ints`. The values 3, 2, and 1 are `ints`, and so the parameters `a`, `b`, and `c` keep them as `ints`. You should not change them to 3.0, 2.0, 1.0.

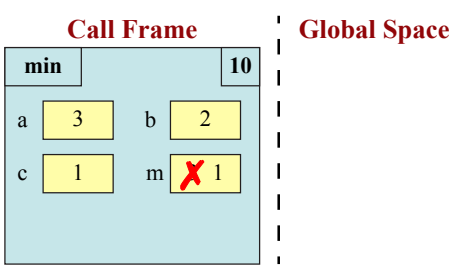
In addition, the assignment statement `d = min(3,2,1)` creates a variable `d` in global space and copies in the return value from the function call. However, this variable is only created when the call frame is erased.

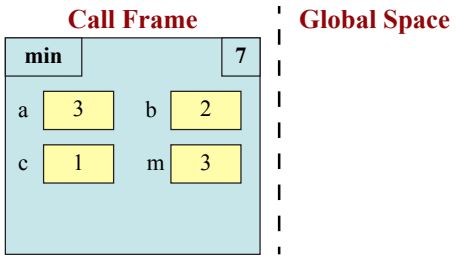
Initial Diagram (Frame Start):

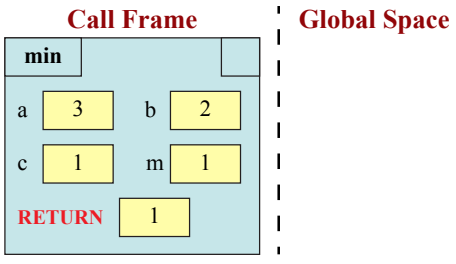


Intermediate Diagrams (Execution):

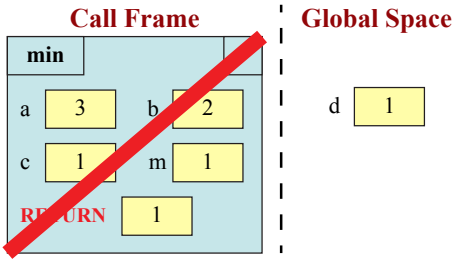








Final Diagram (Frame Deletion):

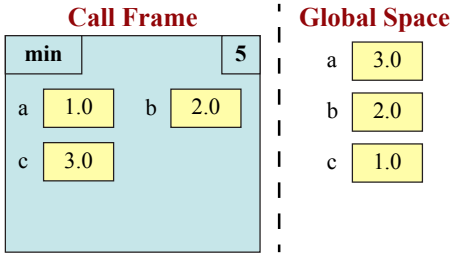


Part B

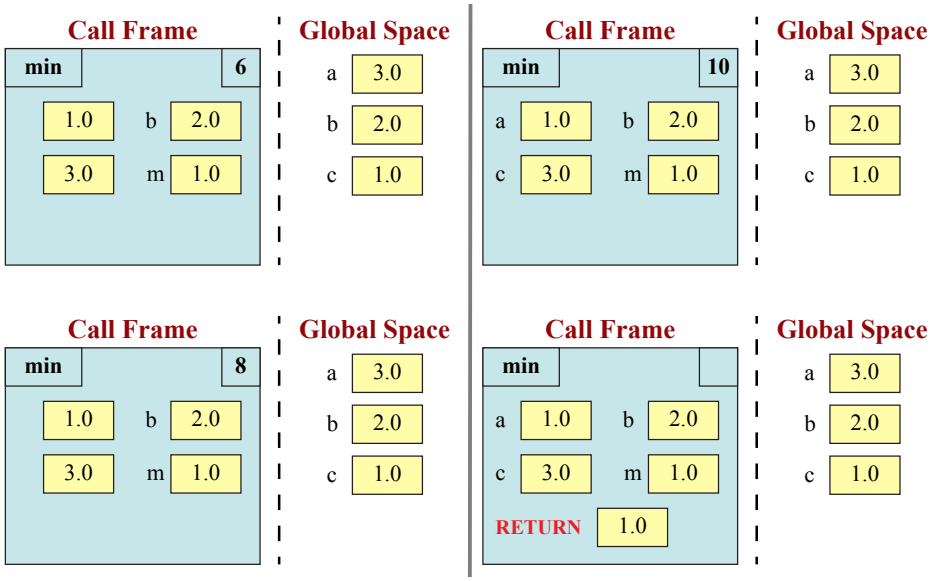
As before, we have to evaluate the function call first, which in this case is `min(c,b,a)`. The *global* variables `a`, `b`, and `c` are different from the *parameters* `a`, `b`, and `c`. When we evaluate the global variables `a`, `b`, and `c`, we see that this function call is the same as `min(1.0,2.0,3.0)`. So the parameter `a` gets the value 1.0, the parameter `b` gets the value 2.0, and the parameter `c` gets the value 3.0. Note that this is different again from `min(3,2,1)` in that 1.0, 2.0, and 3.0 are floats. Do not change them to 1, 2, and 3.

In this function call, the conditional in line 6 is false. Hence we do not execute line 7 this time. In an elif-statement we have to keep checking, so we will check line 8 as well. This is still false, so we skip to line 10. This is all shown below (together with the new global variable `e`).

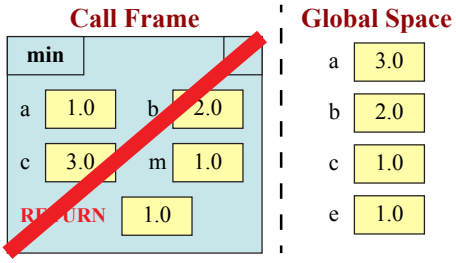
Initial Diagram (Frame Start):



Intermediate Diagrams (Execution):



Final Diagram (Frame Deletion):

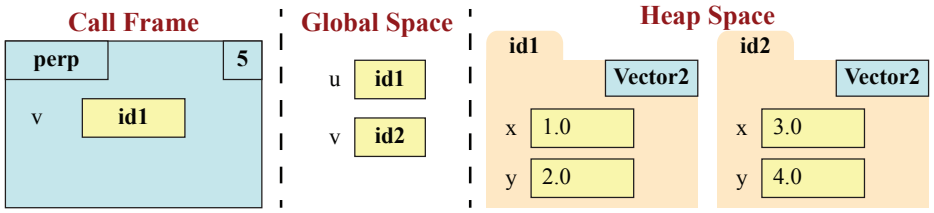


Part C

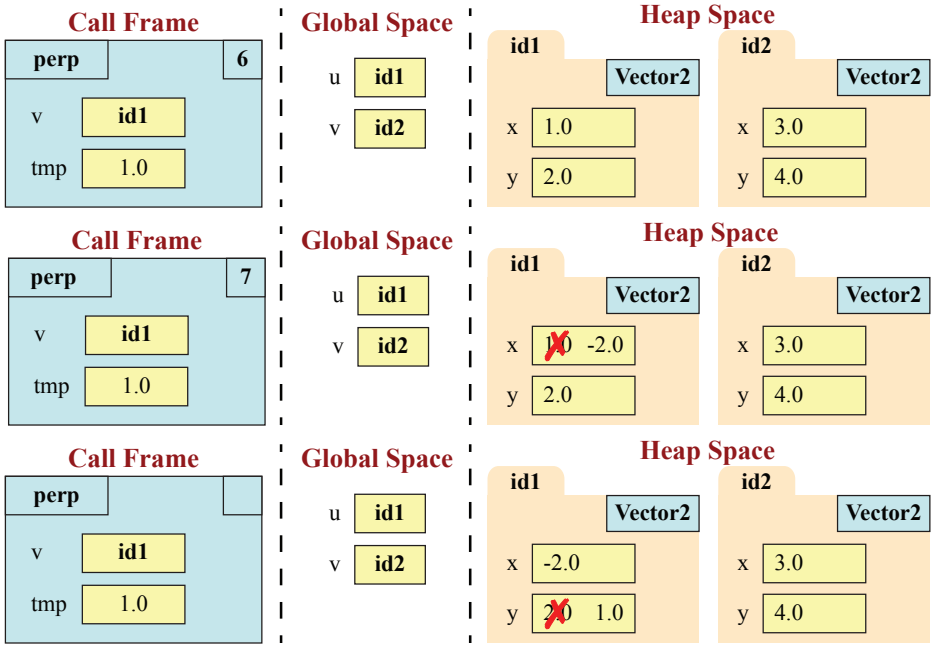
The important thing to understand about this function is that the parameter `v` stores the same name as the global variable `u` (and not the global variable `v`). They both refer to the same folder, which represents a mutable object. The folder itself is in the heap.

This function is a straight-line, with no control flow (e.g. if-statements). This time we have to draw five diagrams for our call frame: the start, the three lines to execute, and deletion. Note that `tmp` stores a non-mutable float value, not a folder name.

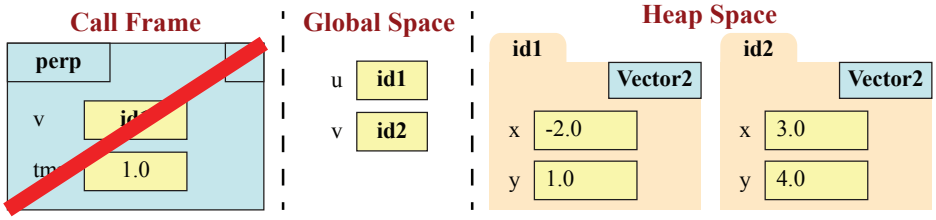
Initial Diagram (Frame Start):



Intermediate Diagrams (Execution):



Final Diagram (Frame Deletion):

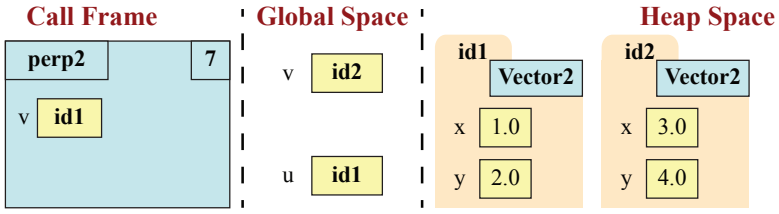


Part D

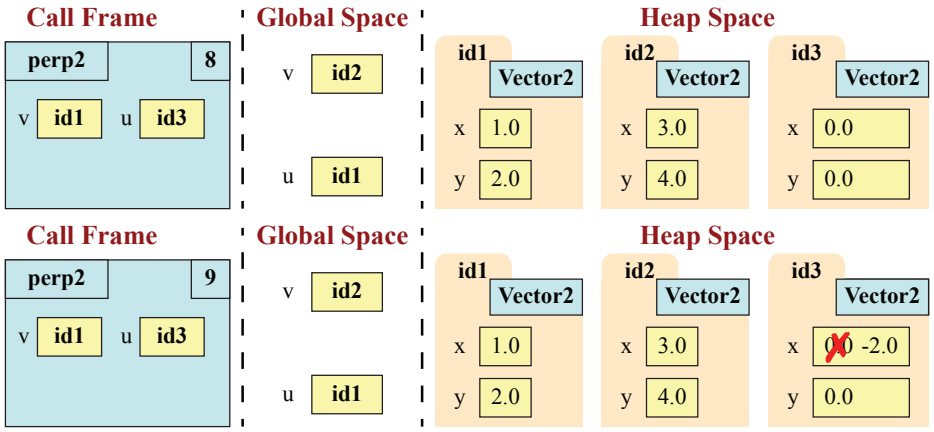
This problem is similar to Part C, except that it is a fruitful function instead of a procedure. It returns the contents of the variable `u`. In addition, we called a constructor function inside this call frame, which results in the creation of a new folder. You may give this folder whatever identifier you wish. We used **id3** in our solution. We only care that you understand that `u` contains this identifier.

Once again, this function is straight line. We have to draw six diagrams this time: the start, the four lines to execute, and deletion.

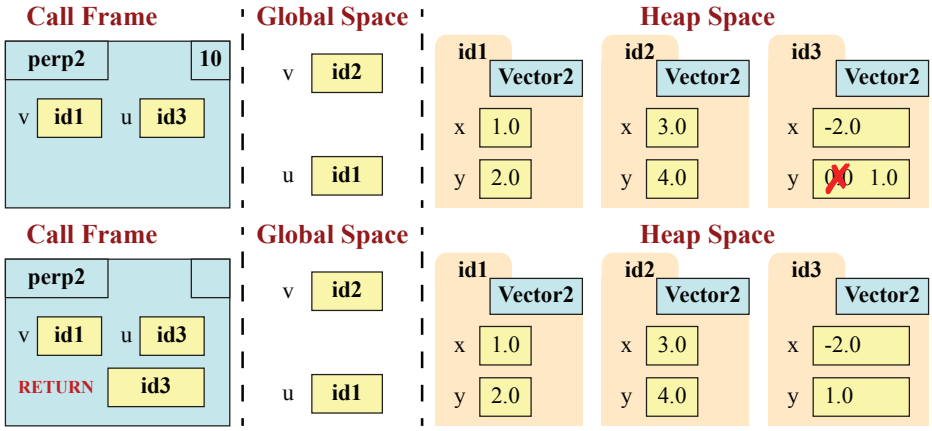
Initial Diagram (Frame Start):



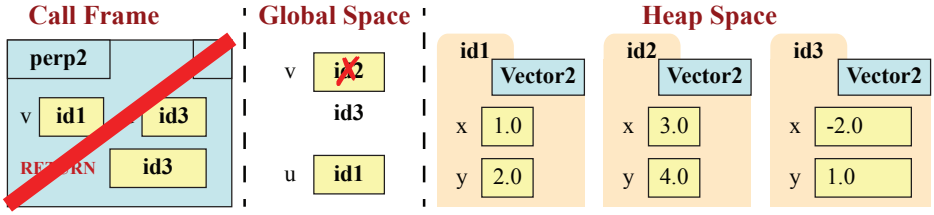
Intermediate Diagrams (Execution):



Intermediate Diagrams (Execution, Continued):



Final Diagram (Frame Deletion):



Part E

For this part of the assignment, there are no pictures – just code. The trick to this question was recognize both of the calls skipped over lines, but they skipped over different lines. Hence you need either an if-elif statement or an if-else statement.

We also see that line 5 is never executed. The else part of an if-else is never executed, so that must be what belongs there. The end result is the following function.

```
1    def dist(x, y):
...          """..."""
12           a = y - x
13           if a < 0:
14               b = -a
15           else:
16               b = a
17           return b
```