CS151 Fall 2012 Lecture 14

Stephanie R Taylor

October 5, 2012

1 Administrative Topics

- We take the quiz
- Stephanie didn't teach today. These notes are from Spring 2011.

2 Lists In Proj 5

2.1 Looping over Lists

Recall the basic structure of a for loop control statement:

```
for item in list:
```

where `item` is the loop control variable and `list` references a list.

There is no rule that the list must contain integers! This list can contain anything. So, we could loop over the items in any list, even the george list.

```
gl = ['George', 3, 26.2, ['The Giant Jam Sandwich', 'Busy Town']]
for gitem in gl:
        print item
```

will result in the output:

```
George
3
26.2
['The Giant Jam Sandwich', 'Busy Town']]
```

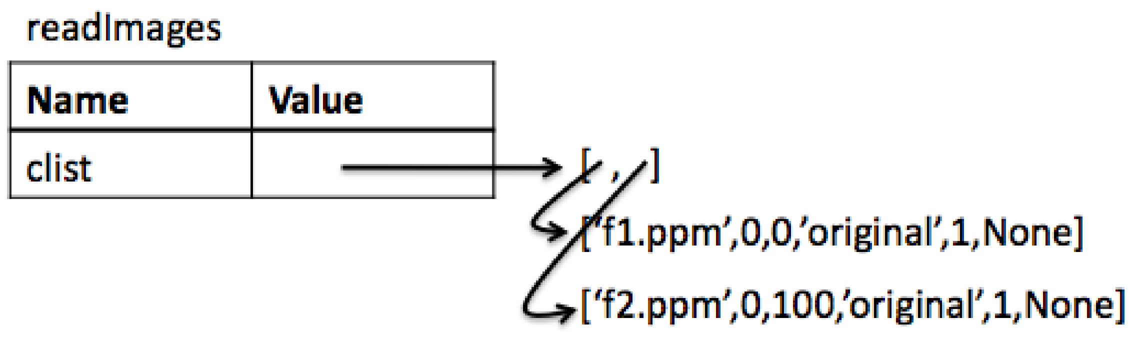
Why would you want to do this? Suppose you are looping through a list of lists. Where each sublist contains information about where to place an image in a collage. Sound familiar?

For example, in lab this week, you wrote `readImages`, which loops over the collage list:

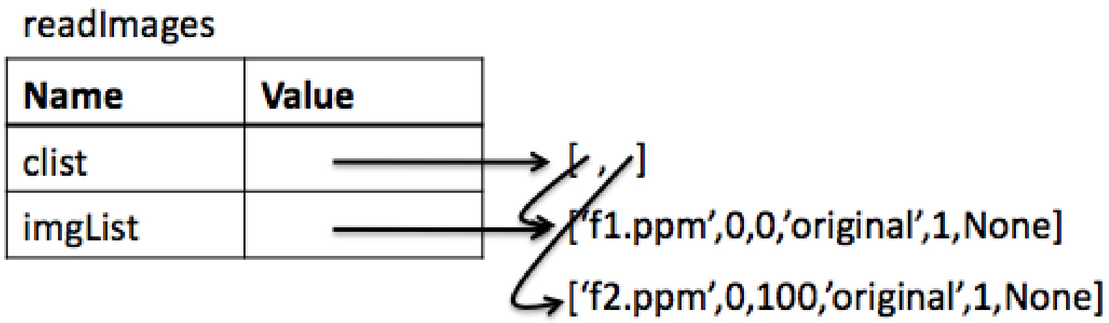
```python
# reads in the files in the collage and stores the
# pixmaps in the list
# clist must have the following format: It must be
# be a list that contains lists about the images.
# the image lists must have this format:
#     filename at index 0
#     x offset at index 1
#     y offset at index 2
#     filter to apply at index 3
#     alpha blend value at index 4
#     Pixmap object at index 5
def readImages( clist ):
    # for each item in clist
    for imgLst in clist:
        # assign to the variable fn the first
        # element in imgList
        fn = imgLst[0]
        # assign to the variable pm, the
        # pixmap returned by reading fn
        pm = graphics.Pixmap(fn)
        # assign to the last element of imgLst, pm
        imgLst[-1] = pm
```

Let's step through this function, examining memory.

Suppose `readImages` is called with a collage list containing information about two images. Then, the symbol table will look like this before any code in `readImages` is executed:


readImages
Name
Value
clist
[,]
['f1.ppm',0,0,'original',1,None]
['f2.ppm',0,100,'original',1,None]

Next, python prepares to execute the loop. The first time through the loop, `imgLst` points to the first sublist of `clist`:

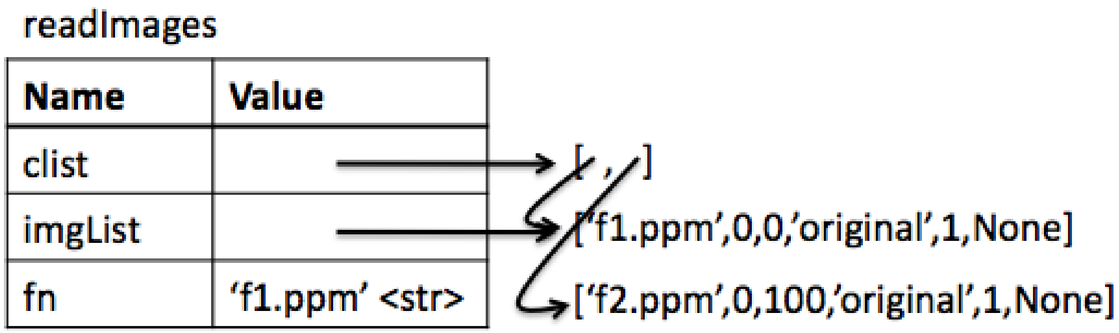


readImages

Name	Value
clist	
imgList	

[,]
['f1.ppm',0,0,'original',1,None]
['f2.ppm',0,100,'original',1,None]

Then, we execute the first line “inside” the loop (line 8). That assigns the file name to `fn`:

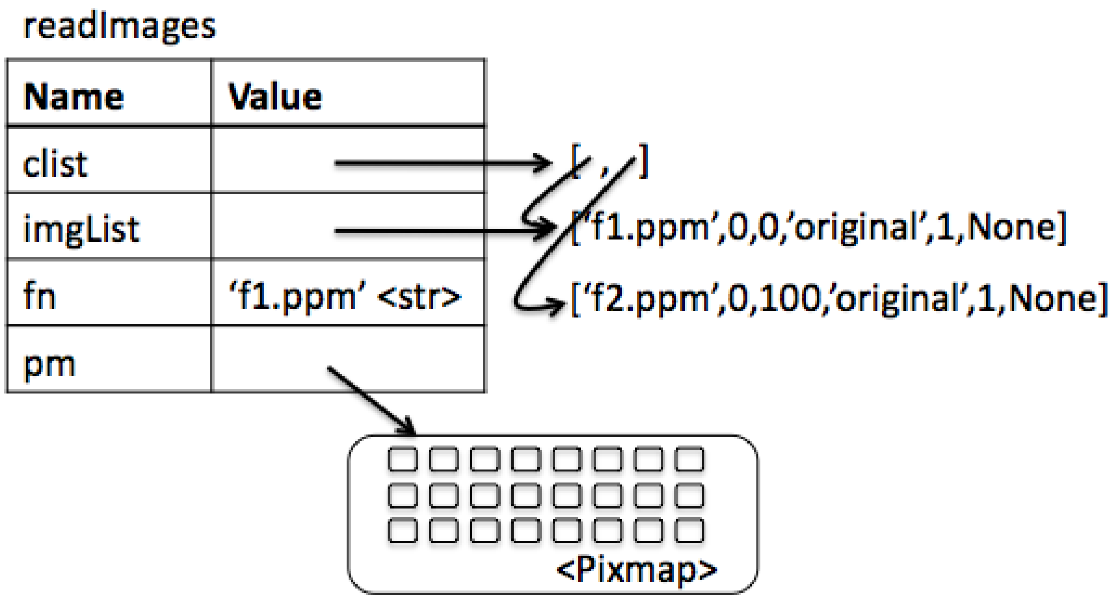


readImages

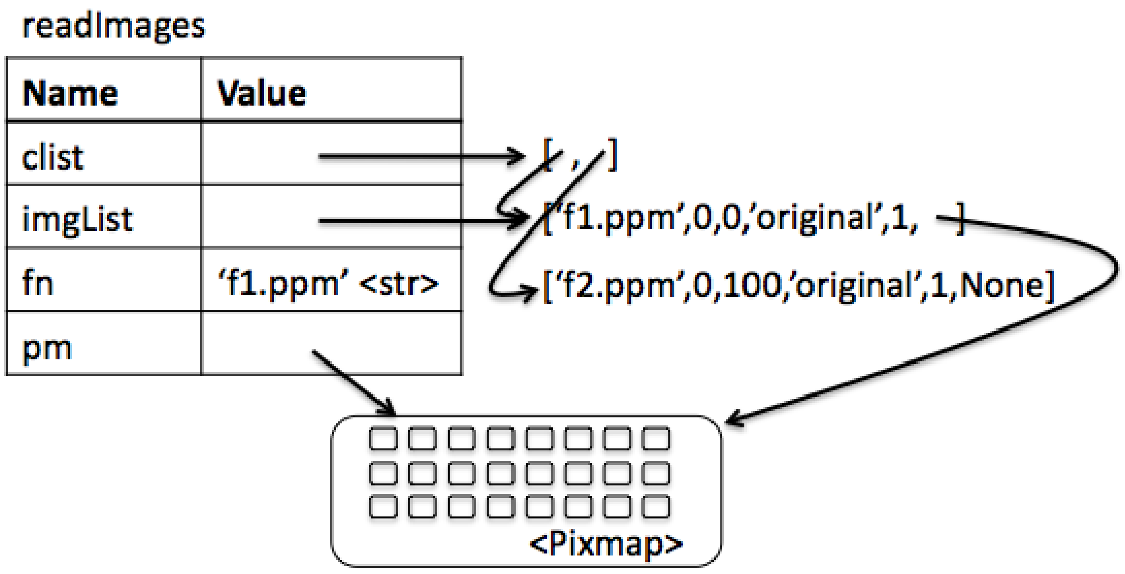
Name	Value
clist	
imgList	
fn	'f1.ppm' <str>

[,]
['f1.ppm',0,0,'original',1,None]
['f2.ppm',0,100,'original',1,None]

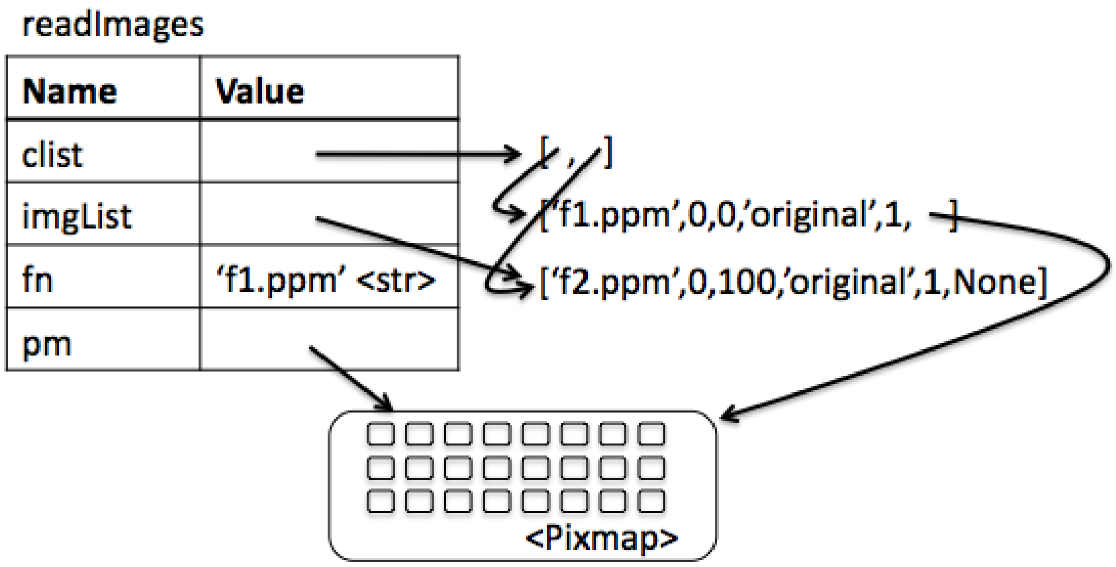
We execute line 11, which causes the graphics package to create a new Pixmap, containing the contents of f1.ppm. The reference is then assigned to `pm`


readImages
Name
Value
clist
imgList
fn
'f1.ppm' <str>
pm
[,]
['f1.ppm',0,0,'original',1,None]
['f2.ppm',0,100,'original',1,None]
<Pixmap>

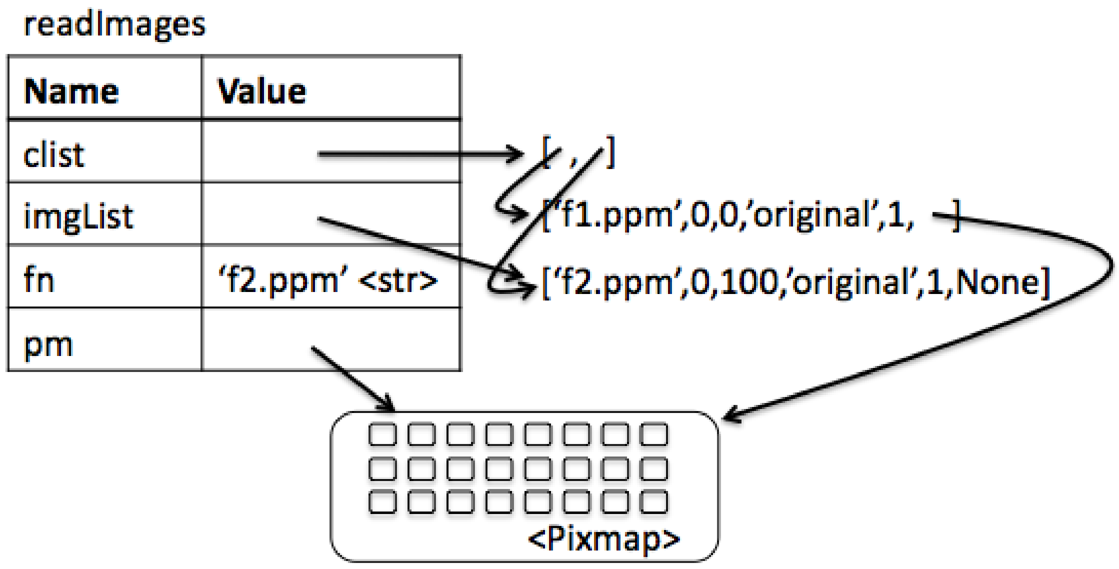
We execute line 13, which replaces the `None` in the first image list with a reference to the pixmap:


readImages
Name
Value
clist
imgList
fn
'f1.ppm' <str>
pm
[,]
['f1.ppm',0,0,'original',1,]
['f2.ppm',0,100,'original',1,None]
<Pixmap>

Python updates the loop control variable `imgLst` to reference the second sublist of `clist`:

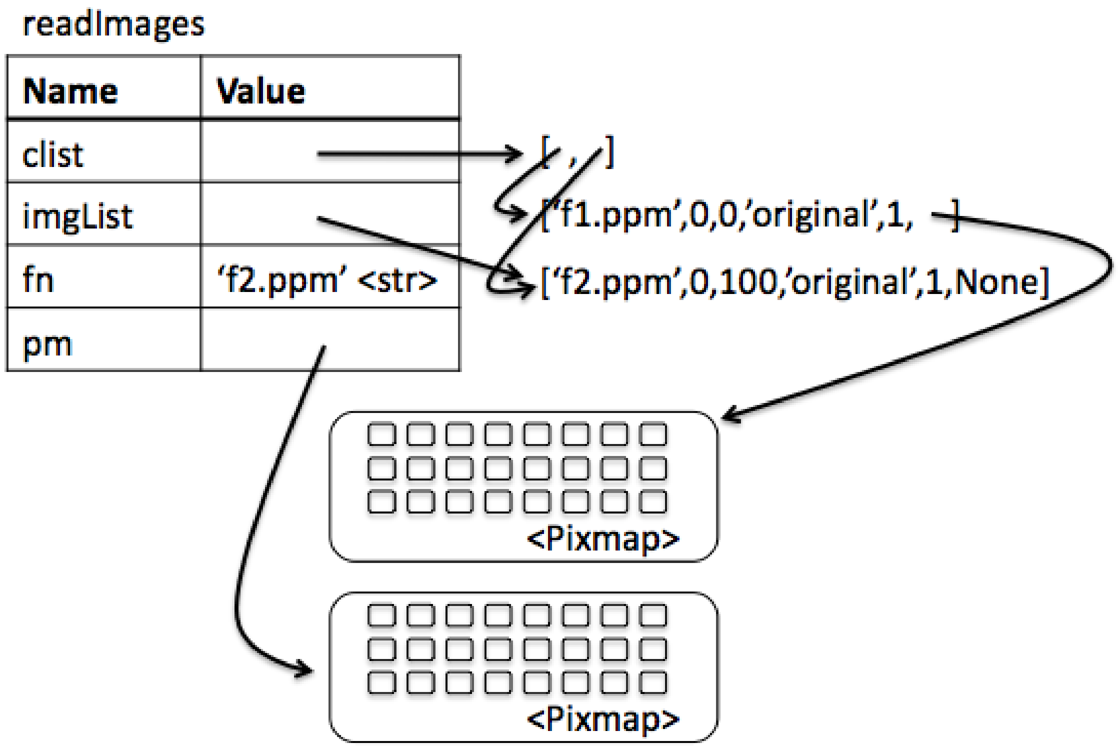

readImages
Name
Value
clist
imgList
fn
'f1.ppm' <str>
pm
[,]
['f1.ppm',0,0,'original',1,]
['f2.ppm',0,100,'original',1,None]
<Pixmap>

Line 8 is executed, changing the value of `fn`:

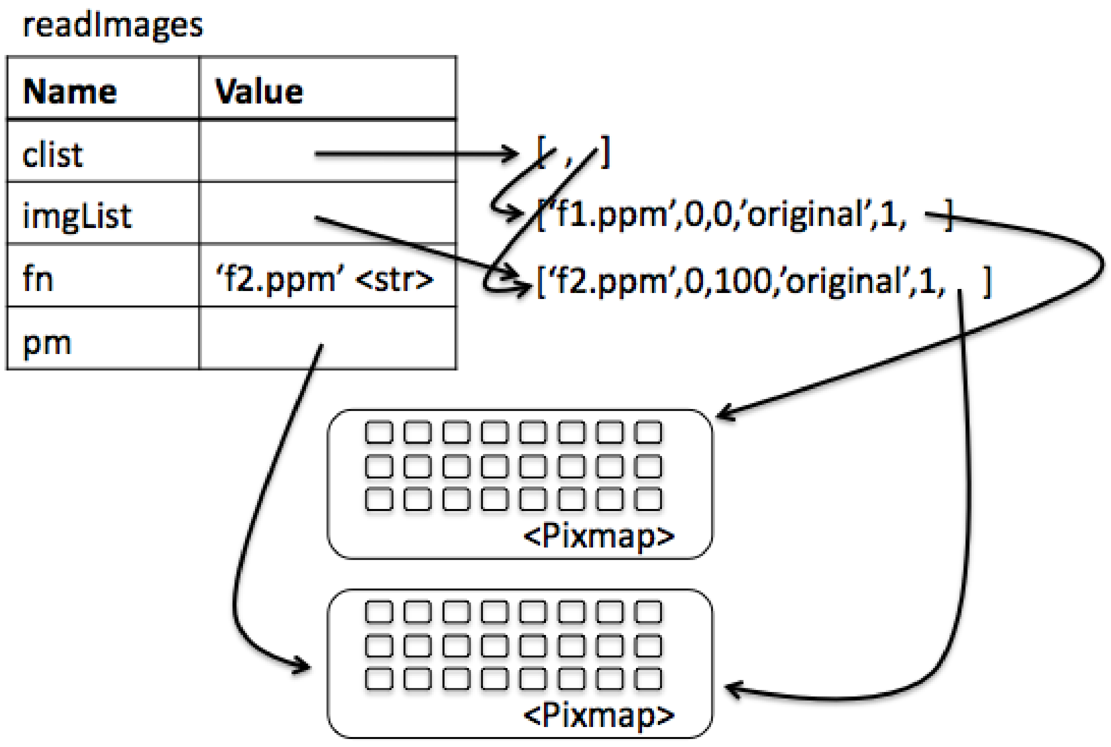


readImages
Name
Value
clist
imgList
fn
'f2.ppm' <str>
pm
[,]
['f1.ppm',0,0,'original',1,]
['f2.ppm',0,100,'original',1,None]
<Pixmap>

Line 11 is executed, which returns a new Pixmap (this time containing the contents of f2.ppm) and `pm` is updated:



Finally, line 13 is executed, and the None in the second sublist of `clist` is updated:



Now the function is ready to return. The caller has its own reference to `clist`, so it will "see" all the changes made to it.

3 Proj 5 Advice

3.1 Triangular Layout

To create the triangular layout when you don't know the sizes of the images ahead of time, use this strategy:

1. Create the collage list, using 0 and 0 for the x and y offsets for all three images.

2. Call `readImages`
3. Update the x and y offsets in your collage list using the size information from the Pixmaps.

For example, if the variable `collageList` references the collage list, and the first image is going to be the top image, then the following snippet of code will place the second and third images “underneath” the first image:

```
1      topheight   = collageList[0][-1].getHeight()
2      collageList[1][2] = topheight
3      collageList[2][2] = topheight
```

Quite a powerful snippet! Quite compact code. Let’s talk about it a little:

Line 1 uses double indexing to access the Pixmap at the end of the first image list. Since `collageList[0][-1]` is a Pixmap object, it has the `getHeight` method, and I can call it using dot notation.

Line 2 assigns the y-offset of the second picture (`collageList[1]` gets to the second image list, and then item 2 (the third item) in it contains the y-offset).

Line 3 assigns the y-offset of the third picture (`collageList[2]` gets to the third image list, and then item 2 (the third item) in it contains the y-offset).

3.2 Random

You can use the `choice` function in the random module to randomly select an item from a list. (It takes as input a list, and returns as output a random item from that list). I do this with my filters. I set up a list of filters.

```
    filters = ['original','rbswap','tint','stripe']
```

Then, each time I want to change/add an filter in the collage list, I simply call `random.choice(filters)`.