CPSC 217 Assignment 3

Due: Monday November 26, 2018 at 12:00 noon
Weight: 7%
Sample Solution Length: Approximately 120 lines, including blank lines, lots of comments and the provided code

Individual Work:

All assignments in this course are to be completed individually. Students are advised to read the guidelines for avoiding plagiarism located on the course website. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

Late assignments will not be accepted.

Submission Instructions:

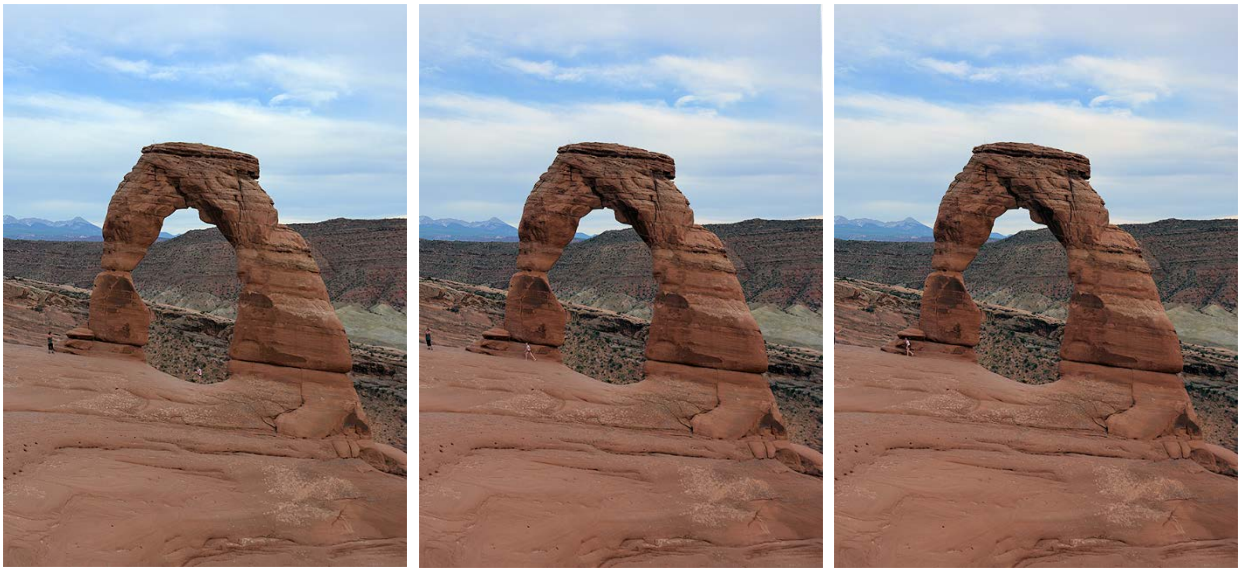
Your program must be submitted electronically to the Assignment 3 drop box in D2L. You don't need to submit `SimpleGraphics.py` or the image files because we already have them (but it won't hurt anything if you include them).

Description

Taking photos of popular landmarks can be frustrating if one wants a picture that doesn't have tourists in it. In fact, it often simply isn't possible to get a shot without a person in it somewhere because of the number of visitors present at the sites. To solve this problem you are going to create a Python program that automatically removes tourists from photos so that only the landmark remains.

You might have read the last sentence and thought "That's impossible!". While that might have been your initial reaction, the reality is that it can be achieved with the right input images. In particular, instead of just taking one picture of the landmark in question, one takes a series of images over some amount of time. During that time tourists come and go but the landmark stays in place. As a result, each pixel in the image will be a pixel from the landmark in most of the images and will only be a tourist in a few (or perhaps even none) of them. This means that the tourists can be eliminated by identifying the pixel color that is most common at each location in the image. While complex clustering techniques can be used to identify this color, we'll simply compute the median of the red, green and blue components of all of the pixels at each location because doing so is reasonably straightforward and gives decent results.

Three images of Delicate Arch (a very popular site in Arches National Park in Utah) are shown below. Notice that there are one or two people in each picture, but that they are in different locations in each image.



Merging these images in the manner described previously generates the following result. In particular, notice that there are no tourists present in the result image because every pixel in the series of input images shows Delicate Arch in at least two of the three images. As a result, the median pixel is a pixel from Delicate Arch.

This assignment is intended to teach you about functions and lists. As a result, you will provide the functionality described previously by writing 5 functions described in the following 3 sections. The starter code I have provided includes a main function that calls 3 of the functions that you will write, and empty implementations of those functions have also been provided.

Part 1: Loading a Collection of Images

All of the data sets for this assignment consist of files named with an abbreviation that describes the subject of the photograph (such as "da" for Delicate Arch or "cb" for Capitol Building), followed by an underscore, an integer, a dot and the filename extension. The integers are sequential beginning with 1.

The `loadImages` function will load all of the images used by the later parts of this assignment. It should begin by prompting the user for the abbreviation that describes the subject of the photograph and the number of photos to load. The abbreviation should be read as a string. You may assume that the abbreviation is correct – you do not need to do any error checking on it. When the number of photos is read you should verify that it is greater than or equal to 3 and less than or equal to 16. If not, your program should display an appropriate error message, close the graphics window by calling `close()`, and quit the program by calling `quit()`. If the entered number is within the required range then your program should load all of the images. The images must be stored in a list, and the list will contain *n* images, where *n* is the integer entered by the user.

Once all of the images have been loaded the graphics window should be resized so that it is twice as wide as the loaded images and has the same height as the loaded images. The width of an image can be determined by passing it as an argument to the `getWidth` function. Similarly, the height of an image can be determined by passing it as an argument to the `getHeight` function.

All of the code described previously should be in a function named `loadImages`. The `loadImages` function doesn't take any parameters. It will return the list of images as its only result. Note that the `loadImages` function doesn't generate any graphical output. You'll write additional functions in later sections to do so. Specifically, Part 2 will discuss how to go about displaying the input images as thumbnails, and Part 3 will discuss how to construct and display the tourist-free image.

My implementation of loadImages was 15 lines of code (plus blank lines and comments).

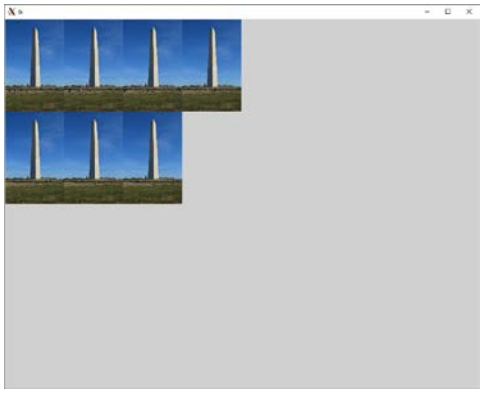
Part 2: Displaying a Collection of Images

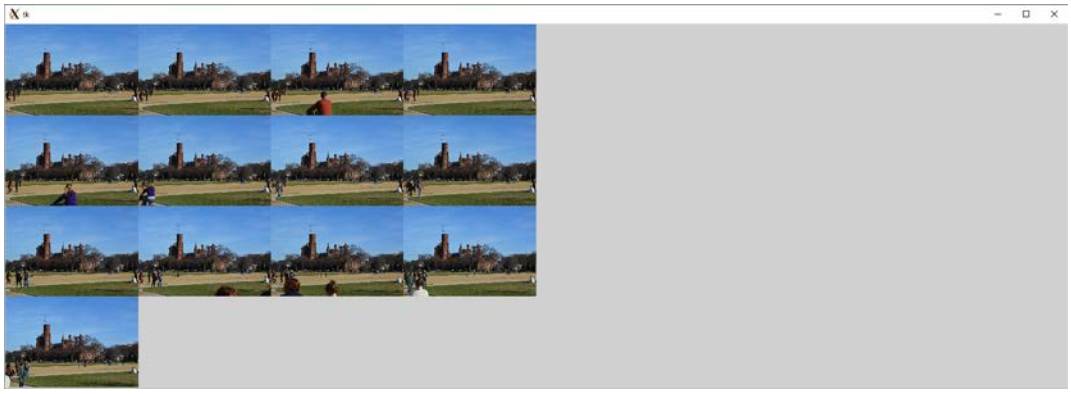
Because there is limited space on the screen we'll display the input images as "thumbnails" that have one quarter the width and one quarter the height (so one sixteenth the area) of the original image. Unfortunately `SimpleGraphics` doesn't have a resize function for images, so you'll have to write a function for creating the smaller image yourself, as described in the following paragraph.

Write a function named `createThumbnail` that takes an image as its only parameter and returns an image as its only result. The body of your function should begin by calling `createImage` to create a new, blank image that is one quarter the width and one quarter the height of the image passed as the only parameter. Then it should use `getPixel`, `putPixel` and nested loops to retrieve every fourth pixel (both horizontally and vertically) from the input image and store them in the thumbnail image. Finally, your function should return the thumbnail image. Note that `createImage`, `getPixel` and `putPixel` require integer arguments for sizes and positions. As a result, you may find it helpful to perform division with the // operator and/or use the `int` type conversion function.

Once you have `createThumbnail` working, write a second function named `drawThumbnails` and use it to create (by calling `createThumbnail`) and display thumbnails for all of the loaded images (which it will take as the function's only parameter). The thumbnails will be displayed on the left side of the window using rows that each contain up to 4 images. Each row should be filled completely before starting the next row. This means that one row will be used if 3 or 4 images are being processed, two rows will be used if 5, 6, 7 or 8 images are being processed, three rows will be used if 9, 10, 11 or 12 images are being processed, and four rows will be used if 13, 14, 15 or 16 images are being processed.

The thumbnail layouts for the Washington Monument (7 images) and Smithsonian Castle (13 images) are shown below. (Your `loadImages` function should have previously resized the window so that it has the same height as the images in the input set and so that it is wide enough to accommodate both the thumbnails and the result image). The `drawThumbnails` function does not return a result.





My implementation of `createThumbnail` was 7 lines of code (plus blank lines and comments) and my implementation of `drawThumbnails` was less than 20 lines of code (plus blank lines and comments). It's fine if your code is longer than mine, but recognize that these are tasks that you should be able to complete without writing a huge amount of code. Also, I'd recommend calling the `update` function (without any arguments) after drawing each thumbnail image so that you can see your program's output as it is generated rather than having to wait for all of the thumbnails to be generated before you can see them.

Part 3:

Now that you have displayed all of the source images it's time to create the tourist-free result. Write a function named `removeTourists` that performs this task. It will take the list of images as its only parameter. The `removeTourists` function does not return a result.

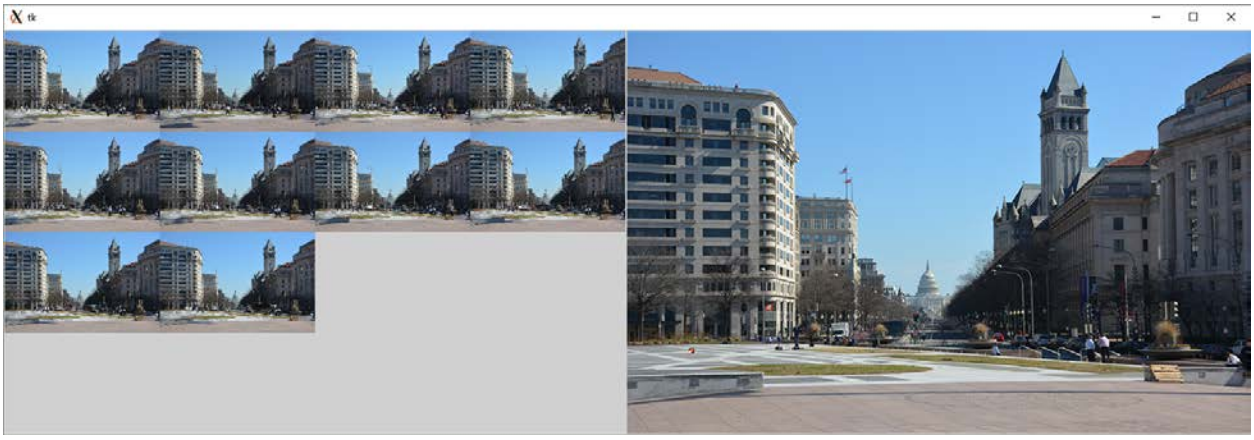
The `removeTourists` function should begin by creating an image with the same dimensions as the input images and displaying it in the right half of the window. (Yes, you can and should display the image before any pixels have been stored in it). Then process every location in the result image using nested loops. Perform the following tasks at each location:

- Build a list that contains the red value for the current location from every source image
- Build a list that contains the green value for the current location from every source image
- Build a list that contains the blue value for the current location from every source image
- Compute the median value of each of the lists above
- Store a pixel with median red, median green and median blue at the current location in the result image

This is a relatively slow process for even modest-sized images because a large number of lists are being processed. As a result, you will likely find it desirable to call `update()` occasionally so that you can see that progress is being made. Calling update for every pixel is too slow so I'd recommend calling it either after each line is complete, or each column is complete, (depending on how you have your loops set up). This means that `update()` will be indented so that it is in the outer of your nested loops but not inside the inner of the nested loops.

Write a function named `median` for computing the median and then call it 3 times – once for the list of red values, once for the list of green values, and once for the list of blue values. The `median` function will take a list of values as its only parameter and return the median of those values as its only result. Recall that we completed an example in class that computed the median of a collection of values entered by the user. You may find it helpful to use some of the code from that example when completing this part of the assignment (and you are welcome to do so).

The expected result for the Capitol Building data set is shown below.



Additional Requirements:

- Include a comment at the top of your file that includes your name, student number and a brief description of the program that you have created.
- You must use the provided main function without modifying it.
- All lines of code that you write must be inside functions (except for the function definitions themselves, any constant definitions, and any import statements).
- You must create and use the functions described previously in this document.
- Do **not** define one function inside of another function.
- You must make appropriate use of loops. In particular, your program should work for data sets that consist of anywhere between 3 and 16 images, and those images should be able to have any size.
- Include appropriate comments for each of your functions. All of your functions should begin with a comment that briefly describes the purpose of the function, along with a description of every parameter and every return value. Functions that do not return a value should be explicitly marked as such.
- Your program must **not** use global variables (except for constant values that are never changed).
- Your program must use good programming style. This includes things like appropriate variable names, good comments, minimizing the use of magic numbers, etc.
- Break and continue are generally considered bad form. As a result, you are **NOT** allowed to use them when creating your solution to this assignment. In general, their use can be avoided by using a combination of if statements and writing better conditions on your while loops.

Image Formats:

- I have provided the data sets as PNG, PPM and GIF files. On my machine the PNG files can be loaded by SimpleGraphics without any trouble, but I have encountered problems with them on other machines. Use the PNG files if you can. If not use either the PPM files or the GIF files. Both of these formats should be able to be loaded by SimpleGraphics on all machines without any trouble. The PPM files are preferable to the GIF files because they are true color images while the GIF files use only 256 colors. However the GIF files are a lot smaller, so they are preferable if you are working on a machine where storage space is at a premium.

Hints:

- The “sm” (stunt man) data set has very small images. As a result they are fast to load, process and display. I’d suggest working with this data set during development so that you can see the effect of changes that you make to your program quickly.
- Don’t want to type the name of the data set and the number of images every time you run your program? Comment out your input statements and replace them with assignment statements that store the values that you would have typed.
- Part 1 needs to be completed first. Parts 2 and 3 can be completed in either order.

For an A+:

Right now the program doesn't do any error checking on the abbreviation entered by the user. It also forces the user to indicate how many images are in the data set when this could be determined by the computer.

Improve the image loading portion of your program so that it only prompts the user for the abbreviation for an image set. Then your program should go on and load all of the images for that set (without requiring the user to enter the number of images). If the user enters an abbreviation for a data set that doesn't exist it should report an error and quit. Similarly, if the number of images for the data set is less than 3 or more than 16 your program should report an error and quit. Note that you must be able to accommodate any set of images, not just the data sets on the course website. As a result, it is **not** acceptable to simply check if the abbreviation is one of "cb", "da", "sc", "sm" or "wm" and then hard code the sizes of those data sets – such a solution will **not** receive any credit. Instead you must use appropriate Python functions / language constructs to determine whether or not the requested images exist and process them appropriately.

If you complete the A+ portion of the assignment, please include a clear, highly conspicuous note indicating such at the top of your submission so that your TA knows to grade it.

Grading:

This assignment will be graded on a combination of functionality and style. A base grade will be determined from the general level of functionality of the program (Does it load all of the images into a list? Does it compute and display the thumbnails correctly? Does it generate the tourist-free image correctly? Etc.). The base grade will be recorded as a mark out of 12.

Style will be graded on a subtractive scale from 0 to -3. For example, an assignment which receives a base grade of 12 (A), but has several stylistic problems (such as magic numbers, missing comments, etc.) resulting in a -2 adjustment will receive an overall grade of 10 (B+). Fractional marks will be rounded to the closest integer.

Total Score (Out of 12)	Letter Grade
12	A
11	A-
10	B+
9	B
8	B-
7	C+
6	C
5	C-
4	D+
3	D
0-2	F