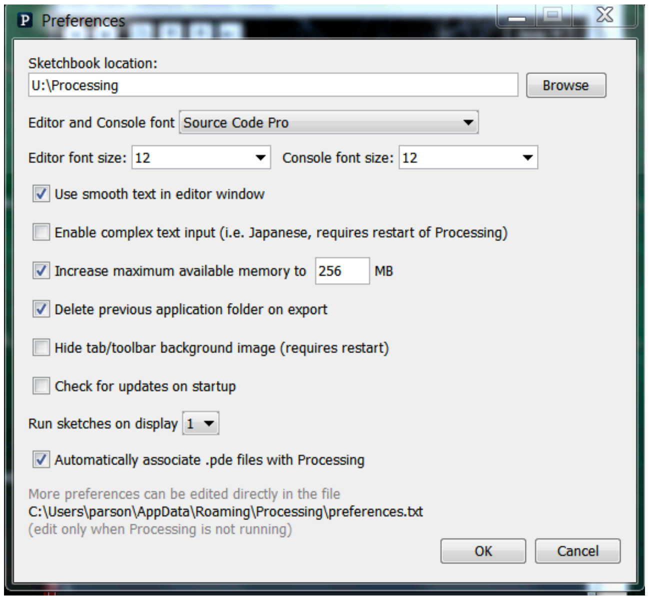
CSC 120 Introduction to Creative Graphical Coding, Spring 2018

Dr. Dale E. Parson, Assignment 2, Implementing and testing a function that plots your avatar.
This assignment is due via **D2L Dropbox <u>Assignment 2 avatarFunction</u>** by **11:59 PM on Saturday March 10 (updated from March 9 due to 3/7/2018 snow day)**.

When using Processing on the Kutztown campus Windows computers, make sure to start out **<u>every time</u>** by setting your Processing Preferences -> Sketchbook Location to U:\Processing. The U:\ drive is a networked drive that will save your work and make it accessible across campus. If you save it to your desktop or the lab PC you are using, you will lose your work when you log out. You must save it to the U:\ drive. If you do not have a folder called Processing under U:\, you must create one using the Windows Explorer. Processing Preferences is under the File menu on Windows.



Sketchbook location: U:\Processing

Editor and Console font: Source Code Pro

Editor font size: 12 Console font size: 12

- [✓] Use smooth text in editor window
- [ ] Enable complex text input (i.e. Japanese, requires restart of Processing)
- [✓] Increase maximum available memory to 256 MB
- [✓] Delete previous application folder on export
- [ ] Hide tab/toolbar background image (requires restart)
- [ ] Check for updates on startup

Run sketches on display 1

- [✓] Automatically associate .pde files with Processing

More preferences can be edited directly in the file
C:\Users\parson\AppData\Roaming\Processing\preferences.txt
(edit only when Processing is not running)

Open your working avatar from assignment 1 in Processing and **save it as sketch <u>avatarFunction</u>**. You will turn in file **avatarFunction.pde**. If you lost points because of omissions or bugs in assignment 1, you must fix them as part of this assignment.

Review my example avatarFunction2018 linked on the course page next to assignment 2.[1]

[1] http://faculty.kutztown.edu/parson/spring2018/avatarFunction2018.txt

REQUIREMENTS:

1. Create a function display(…) that takes a minimum of four parameters, namely an **X location**, a **Y location**, a float **scaling** parameter, and a body-part-wiggling parameter, similar to the **avx**, **avy**, **avscale**, and **avwiggle** parameters in my display() example function. You may add other parameters for color changes or other properties as you like. This function **DEFINITION** must go after the draw() functions closing curly brace.
2. Within function display(…), call **pushMatrix** as the first statement, **translate** as the second, **scale** as the third, and **popMatrix** as the last. Use the appropriate function parameters for translate and scale. Note that scale can take 2 parameters, one for X and another for Y; you could pass two scale parameters and use them in your function. The purpose of starting your function in this way is to position and scale your avatar differently without changing the avatar plotting code itself. The purpose of putting pushMatrix at the start is to record the coordinates in effect when your function is called from draw(), which is drawing the scenery; the call of popMatrix at the function's end is to restore those scenery coordinates. You could also add a rotateAngle parameter and a call to **rotate** for 1 bonus point.
3. Move all of your code from assignment 1 that actually draws the avatar into your display() function, after the above geometric transforms and *before* popMatrix, making any changes you find necessary to fit your avatar into this function. **Change all global variable names within the function that have function parameter counterparts to the parameter names**. Do not move scenery code into the function.[2]
4. Place a call to display(…) at the original location in the assignment 1 code, and ensure that the avatar still draws correctly. This call **replaces** the original avatar-drawing code within draw().
5. Add at least one additional call to display(…) within draw(), using different values for the other parameters. You can plot multiple avatars at the same time, or avatars at different scales at different times. Your initial avatar needs to be mobile, but you can make additional avatar(s) stationary by passing constants as their arguments; or, you can use variable arguments for your additional avatar() calls.
6. Use an "if" construct at least once in your sketch to make a decision, for example to reverse direction. Using "else if" or default "else" portions is optional.
7. You must write at least one "while" or "for" loop that has some visible effect. My avatarFunction2018.pde use a "for" loop to plot multiple avatars at the same time. What you do with the repetition of a "while" or "for" loop is up to you.
8. All of the assignment 1 requirements for motion, moving body part, foreground & background scenery, etc. are still required. Assignment 2 adds to assignment 1 requirements.
9. Make sure to update the documentation at the top of your sketch. Add comments where useful to make the code clearer to understand.

Grading: Each of the above 9 steps is worth 10% each (90% total). Maintaining assignment 1 functionality earns 10% (or adding it if it was missing in assignment 1).

Drop avatarFunction.pde into D2L Assignment Dropbox Assignment 2 avatarFunction by 11:59 PM on Friday 10.

[2] Note that code to update global variables, such as my *avatarX*, *avatarXspeed*, *legX*, and *legXspeed* variables, must stay **outside** the display(...) function. Otherwise, every call to display(...) will update these variables. My code updates these variables immediately after the last call to display(...). There is no point in updating display(...)'s parameters or local variables just before the function ends, because parameters and local variables cease to exist after a function call completes.