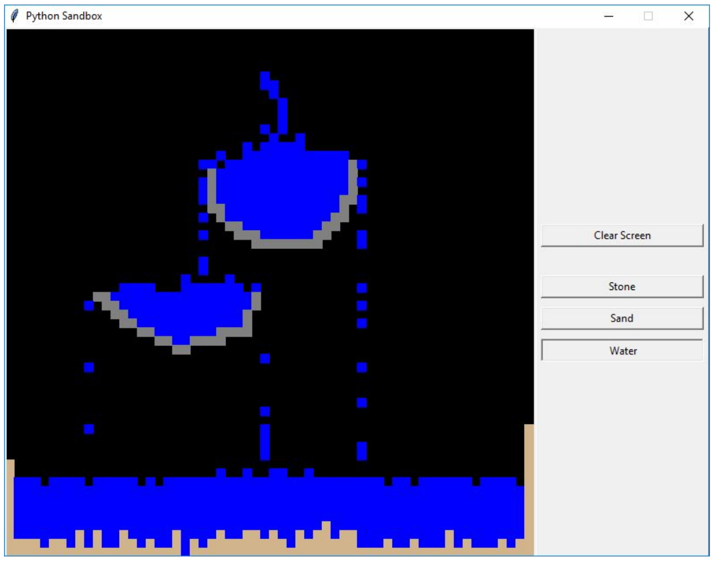
CS 101 - Assignment 5 - Sandbox
Spring 2018
Algorithm Due : **March 18th, 2018**
Program Due : **March 25th, 2018**
All work submitted must be your own.
Deliverables :

You only need to submit your solution.
You should not submit the gravity_mod.py module.
You must use functions to modularize your work.
You should use exception handling where necessary as well.

Sandbox

You are going to create a graphical simulation of 3 different materials. Rock, Sand and Water. They all behave differently, a block of Rock will hover wherever the user places it. (We assume it's made of Unobtanium from Avatar). Sand falls straight down, it only stops if it hits rock or the bottom of the window. It will fall through water. Water will fall straight down, but if there is another block below the water or it is at the bottom of the window, then it will flow randomly right or left if there is a block open. If it is "blocked" in on all sides, it will not move.



Gravity Module

You won't have to worry about creating buttons and organizing the screen. The gravity module will do all of that for you. This is a user created module and will need to be in the same directory as your solution. You should not change or submit it. Only submit your solution. You will start out with a solution given to you that has a basic loop that just draws a rectangle. There are really only 4 functions and methods to deal with.

```
import gravity_mod as gfx
```

```
# Creates a window 100 wide, and 100 tall
win = gfx.GravWindow("Newton was right")

# Draw on the screen
win.draw_rect(10, 10, 40, 50, "Blue")
win.draw_screen()
```

Let's take a look at that line by line.

```
import python_graphics as gfx
```

This imports the python_graphics module as gfx. This allows us to give the module an alias so we can just type gfx<dot> instead of python_graphics<dot> each time we want to call a function in the module.

```
win = gfx.GravWindow("Newton was right")
```

This line creates the new window. You call the Window function to create the new UI window and assign the reference of that window to a variable, in this case win. The GravWindow function has two arguments; the first is the name of the window, this will be put in the title bar of the window. The title is an optional argument and will default to Python Graphics. The title in the above screenshot isn't visible since it is too small of a window. You can also assign a background color, but it will default to a black background if you don't specify one, which is probably the best choice for color for this assignment.

```
win.draw_rect(10, 10, 40, 50, "Blue")
```

Draws a blue rectangle where the upper-left corner is at 10, 10, and the lower right is at 40, 50. In

```
win.draw_screen()
```

Any rectangles or text you draw on the screen will not be displayed until you call the draw_screen method. If you call draw_screen again, then it will erase your screen and it will be empty. (so you need to draw everything on the screen each time you want to show something.)

```
events = win.get_events()
```

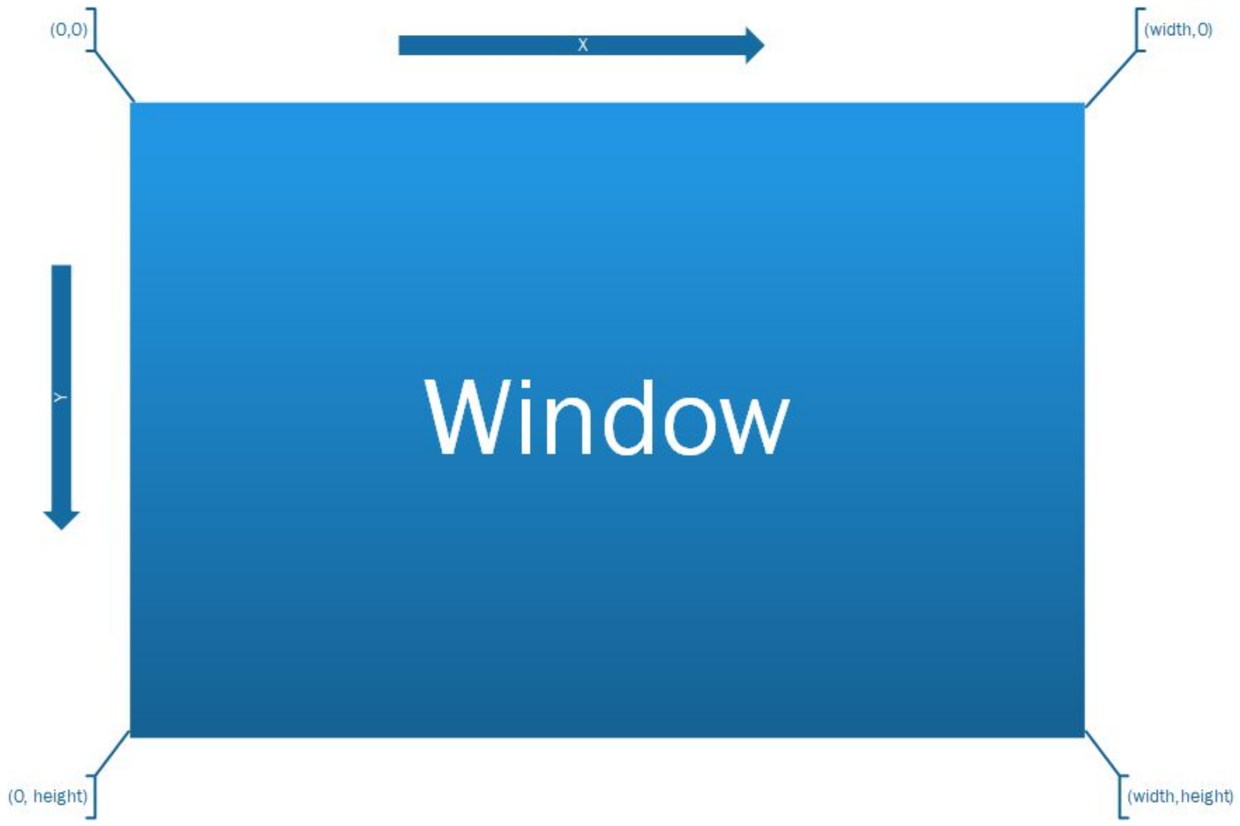
This will return a list of all the events that have happened since the last call. The events can be a single string, or a tuple (so you need to test each element to see what type it is). If it is a string, there is only one command, “CLEAR” which means the user pressed the clear screen button and all blocks should be removed from the screen so they can start drawing again. If it is a tuple, then it will contain 3 items in the tuple; the material type (STONE, SAND or WATER), and the x and y coordinate where the mouse event occurred at.

Sample Events

```
[
('STONE', 397, 399),
('STONE', 397, 378)
'CLEAR']
```

Screen Coordinates

A computers coordinates aren't quite like the cartesian coordinates that you are used from math class. The screen coordinates start with 0, 0 at the top left and increase the x values going right, and increase the y values going down. So a point at (50, 100) would be 50 pixels right of the origin at the top left of 0, 0 and 100 pixels down from the top left 0, 0.



Time Module

We can use the sleep function in the time module for a delay. (You may need that for debugging.

```
>>> import time
>>> start = time.time()
>>> print("Before sleep")
>>> time.sleep(1.2)
>>> end = time.time()
>>> print("The program slept for ", end - start, "seconds")
```

time.sleep(1.2) will pause our program for 1.2 seconds. In the above example our elapsed time should be approximately 1.2 seconds. You won't want to keep drawing the screens over and over. It would end up making the program run too fast. You may want to try to sleep 0.1 seconds before the next frame of the game.

Specification and hints

- You are provided a blank solution. Use that for your program and submit only that. You do not need to submit gravity_mod.
- Show the play screen with a custom title. The default size should be used, so you don't need to change it.
- The draw area is 600 x 600. That's 600 pixels by 600 pixels. Each block size is 10 pixels by 10 pixels. Therefore, it is 60 blocks wide, and 60 blocks tall. The coordinates given will be in pixels, so you have to convert that to which block the click occurred in.
- Drawing the block will take pixel coordinates. Block at 0, 0 will have the top left at 0, 0 and the bottom right at 9, 9. Block 1, 0 would then have the top left at 10, 0.
- When the user draws rock, sand or water it should be put onto the screen if there is not a block currently after that location.
- The screen information should update each cycle.
    - Rock will never move, so you may want to start with the rock.
    - Sand,
        - If there is an open block below it, then it should fall down into the next spot, unless its at the bottom of the screen.
        - Otherwise, it stays in the current position
    - Water
        - If there is an open block below the water, it should fall down, unless it is at the bottom of the screen
        - Otherwise, it should randomly move to an open spot to its left or right
- Once a block has been moved during a cycle it should not move again.
- If the player hits clear then all the information on the screen should go away.
- Start slow, try different ideas and theories out slowly and build from there.

Material	Color
Rock	"Gray"
Sand	"Tan"
Water	"Blue"

Point Breakdown - May be modified as needed

Points	**Requirement**
5	Header

25	Readability, variable naming, comments, structure
7	Draw Rock on screen when selected and no block under cursor
7	Draw Sand on screen when selected and no block under cursor
7	Draw Water on screen when selected and no block under cursor
7	When updated Rock stays in current position
13	On update, sand falls directly down if there is no block underneath it and it is not already at the bottom
15	On update, sand water falls directly down if there is no block underneath it and it is not already at the bottom. If water cannot fall down, then it randomly moves left or right if there is nothing currently present.
7	Clear will remove all current material from screen and let the user start over.
7	Error is caught when the user closes the window, output thanks for playing and the game can end.

50 points off for programs that crash on expected input.

Two-Dimensional Lists

How you keep track of a 2-dimensional grid is up to you. There are many different ways to represent the screen. One idea is a list of lists. It is a list that contains' other lists in it. Each sub list is a row in the grid. If we need a grid that is 4 columns wide by 5 rows.

```
grid = [    [  0,  1,  2,  3],
            [  4,  5,  6,  7],
            [  8,  9, 10, 11],
            [ 12, 13, 14, 15],
            [ 16, 17, 18, 19]]
```

grid[2][1] would then reference row 2 and column 1.

References

1. Tkinter - https://wiki.python.org/moin/TkInter
2. Video of Simulation in Action. https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=ee2e0046-2173-4e31-bb9b-a86f012bb209