Modular Arithmetic & Cyclical Counting

We've been looking at expressions of the form, `<exp> % N` where `<exp>` is some numerical arithmetic involving a combination of addition, multiplication, and subtraction and `N` is some positive integer. When working on drawing different dots in class, we encountered the expression `(y+5) % 20` where y was the coordinate we needed to specify which dot to draw.

One of the key observations we've made about these expressions is that **the result is always an integer between 0 and (N-1)**. When animating our dots, we decided that we need the computer to effectively count in a cycle: 0, 5, 10, 15, 0, 5, 10, 15, and so on. Modular arithmetic exactly solves this problem for us! We can start `y` at 0, then increment `y` by 5 modulo 20 and get the desired cycle.

```
y = 0
y = (y + 5) % 20
```

Game Loop with animation

After lab, you had a game loop with logic similar to what we might have if we only wanted to draw one of our dots:

```
while True:
  # check for QUIT

  # draw
  DS.blit(PC,(40,25),(5,0,5,5))

  # update the display
  pygame.display.update()
```

During the drawing portion of the loop, you direct the program to draw the same thing every time. This is made evident by the fact that everything involved in specifying what gets drawn and where it's drawn is constant. We call this a *static* system. It doesn't change as the program progresses.

For animation, we need a dynamic system in which we draw something different from one step of the loop to the next. To accomplish this we use variables to track and specify the dynamic behavior. For drawing our dots in class, we needed a single variable to tell us the y coordinate of the dot to be drawn. All other parts of the statement stay the same regardless of which dot in your column you want drawn. This leads to the following game loop logic:

```
# declare and initialize the variable
y = 0

while True:
  # check for QUIT

  # update variables
  y = (y+5) % 20

  # Draw the dot according to the variable
  DS.blit(PC,(40,25)(5,y,5,5))

  pygame.display.update()
```

When this game loop is run, then value of y will cycle through 0, 5, 10, and 15 and result in the player seeing the dot change over time. In your game animation, you're replacing dots with player drawings and the result is a program that cycles through the images that correspond to the character walking in one direction.

Vertically "Slicing" A background

Do create the background animation, we need to imagine drawing a vertical line down the middle of the background and then drawing the right portion on the left side of the screen and the left portion on the right of the screen. We started exploring this with our 75 x 50 background by drawing the line at x=55. This resulted in the following rectangles:

```
(0,0,55,50) # left side
(55,0,20,50) # right side
```

From here we start relating these concrete values to some key attributes of our problem, namely: `x`,`SCREEN_W`, and `SCREEN_H`. This led to the following generalization.

```
(0,0,x,SCREEN_H) # left side
(x,0,SCREEN_W-x,SCREEN_H) # right side
```

Using the above, we can take any value of `x` between 0 and `SCREEN_W -1`, any `SCREEN_H`, and any `SCREEN_W` and split the background image/surface in to two portions.

Swapping Sides as we Blit

Now that we can split the background, we can turn towards drawing the left side of the background to the right of the screen and the right to the left. This means figuring out the top-left coordinates for the two regions.

Placing the right to the left is the easy one. The left side of the background always has its upper-left corner at the origin `(0,0)`. This means we blit the right side of our background to `(0,0)` of the display surface in order to place it on the left side of the scene.

Figuring out the placement of the left side of the background is a bit trickier. The y coordinate is still 0. We're lining up all the background images with the top of the scene. The x coordinate takes a bit of thought. The left side portion of the background needs to get placed into the display surface after the right side portion (which is now on the left side of the display). Returning to the concrete example, we noted that the right side of the background had a width of 55. This means we blit the right side of the background at 55 in the scene, i.e. after the left side of the background. To generalize the example we observe that 55 was our value for `x`. This means to blit the left portion of the background into the right portion of the scene we blit it to `(x,0)` in the display surface.

Animation

Now that we can deconstruct the background into two portions and swap them as we place them into the scene, all that's left is to vary `x`, the location of the vertical slice over time. The first key observation is that `x` must stay between 0 and `SCREEN_W-1`. This is accomplished by doing all our arithmetic modulo `SCREEN_W`, just like we did when animating our character!

```
x = (x+??) % SCREEN_W
```

Now we need to decide how to increment `x`. The good news is that you can replace the `??` with any value you want as there's no need to slice the background on a specific increment like we did with the character images. By adding values to `x`, we'll get the effect or left to right movement. By subtracting values from `x`, we get right to left movement. The larger the value, the faster the movement. Play with it and find something you like!.