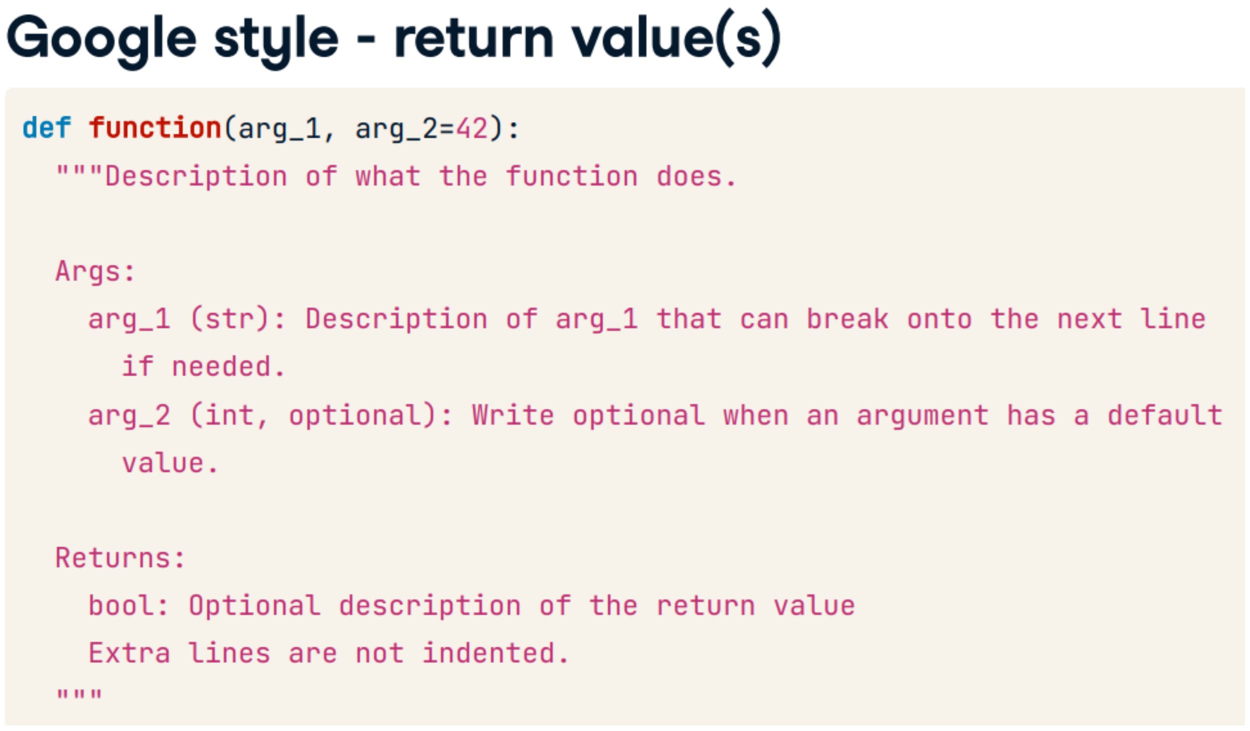
GR 5072 Activity

1. Your task is update the **t_test**() function from last week's class activity in the following ways:
   - Update the function so that it can take in multiple numeric and binary variables (rather than just one of each) in the form of list objects with one or more elements each
     - This week, a key focus is on for loops! So, make this update using for loops
     - Rename the two input arguments as "num_vars" and "bin_vars"
   - For the "bin_vars" argument, update the function to check that all of these variables are binary. Make sure that it can take any variable as long as it has two categories – including string variables! If there is a single variable which is not binary, stop executing the function and return the following *AssertionError*: "All bin_vars must be dichotomous!"
     - *Hint1*: Look at the week05.ipynb file, in the section "Stop functions and send an error" for how to raise an *AssertionError*
     - *Hint2*: Use a for loop to loop through the bin_vars, checking if they are all binary or not. I suggest creating a Boolean pd.Series object that checks these variables one by one, and then at the end you can use the all() function to see if there are any non-binary variables
   - Based on the previous bullet point, you need to generalize how you are subset the data to create the group1 and group2 objects so that any binary variable works, not just 1/0 variables
   - Add a docustring to this function following Google's style, a style which I've pasted below. Make sure you keep the format and naming consistent with the image below, so the spacing should be identical. Make sure have sections for Args, Returns, Raises (here, put the check we performed above to make sure all bin_vars inputs are truly binary).



Google style - return value(s)

```
def function(arg_1, arg_2=42):
  """Description of what the function does.

  Args:
    arg_1 (str): Description of arg_1 that can break onto the next line
      if needed.
    arg_2 (int, optional): Write optional when an argument has a default
      value.

  Returns:
    bool: Optional description of the return value
    Extra lines are not indented.
  """
```

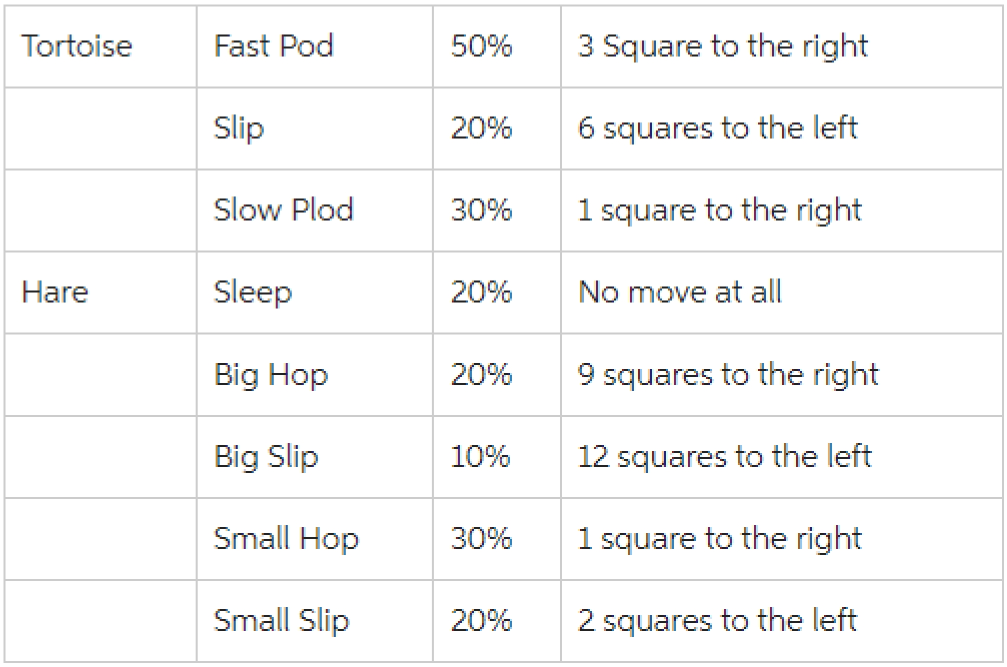
2. Create a new variable called *ged_cats* which recodes the *ged* variable in the following way: if *ged* = 1 then *ged_cats* = “ged”, else *ged_cats* = “no ged”. With this new variable created, call the updated **t_test**() function with the following inputs:
   - num_vars=[“income_log”, “post_sec_edu”], bin_vars=[“ged”, “ged_cats”, “female”]

   Does your function appear to be working? Do you get the same answers when the *ged* and *ged_cats* variables are compared to the same numeric variable? If not, fix your function.

3. Call the updated **t_test**() function with the following inputs:
   - num_vars=[“ income_log”, “post_sec_edu”], bin_vars=[“ged”, “BYRACE”]

   Does your **t_test**() function stop executing and return the following *AssertionError*: “All bin_vars must be dichotomous!”? If not, fix your function.

4. In this problem you will re-create the classic race of the tortoise and the hare. You will use random-number generation to develop a simulation of this memorable event. Our contenders begin the race at “square 1” of 70 squares. Each square represents a possible position along the racecourse. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground. Each animal has an opportunity to make one move at a time depending on their luck:



Tortoise	Fast Pod	50%	3 Square to the right
	Slip	20%	6 squares to the left
	Slow Plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big Hop	20%	9 squares to the right
	Big Slip	10%	12 squares to the left
	Small Hop	30%	1 square to the right
	Small Slip	20%	2 squares to the left

Use variables to keep track of the positions of the animals (i.e., position numbers are 1-70). Start each animal at position 1. If an animal slips left before square 1, move the animal back to square 1. Being the race by displaying:

BANG ! ! ! !

AND THEY’RE OFF ! ! ! !

For each turn or step the animals get to take, generate the percentages in the preceding table by producing a random integer *i* in the range 1<= *i* <= 10. For the tortoise, perform a “fast plod” when 1 <=*i* <= 5, a “slip” when 6 <=*i* <=7 or a “slow plod” when 8<=*i*<=10. Use a similar technique to move the hare.

For each tick of the click (i.e., each repetition of a loop), print a 70-position line showing the letter **T** in the tortoise’s position and the letter **H** in the hare's position. This “line” should be white spaces, other than the starting gate on square 1 which you can print “_” and the finish line where you should print “|”. Occasionally, the contenders land on the same square. In this case, the tortoise bites the hare, and your program should print **OUCH!!!** beginning at that position. All print positions other than the starting position, the finish line, the **T**, the **H** or the **OUCH**!!! (in case of a tie) should be blank.

After printing each line, test if either animal has reached or passed square 70. If so, print the winner and terminate the simulation. If the tortoise wins, print **TORTOISE WINS!!! YAY!!!** If the hare wins, print **Hare wins. Yawn. . .**

If both animals win on the same clock tick, you may want to favor the tortoise (the "underdog"), or you may want to print **It's a tie.** If neither animal wins, perform the loop again to simulate the next tick of the clock.