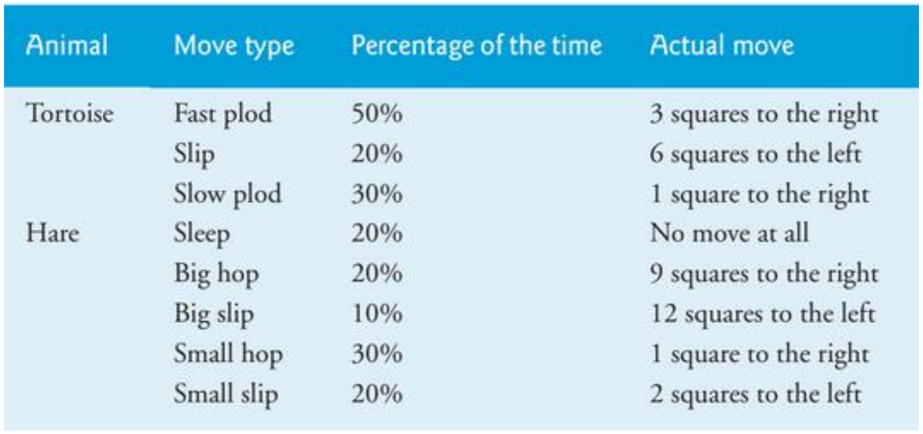
Week 2 and 3 – (Lists, Tuples, Functions, Dictionaries, Sets)

1- **(Play the Multiplication Game)** Write a script that lets the user play a multiplication game. To begin, the game randomly selects two numbers between 1 and 10. It displays the selected numbers and asks the player for the product of both numbers. If the player answers correctly, they are asked if they want to play again. This cycle is continued until the player makes a mistake or stops the game. Modify the previous to count the number of correct multiplications. At the end of the game the total number of correct exercises is displayed on the screen. Extra challenge: You can extend this exercise further by adding a computer opponent to the game. Write a function that automatically calculates the product of two randomly selected numbers. Add some intelligence to the opponent by implementing smart rules in the function that allow the opponent to make an occasional mistake. The human player and his opponent take turns to calculate a product. The game ends when the player indicates that they do not want to play anymore. The winner of the game is the player with the highest number of correct calculations.

2- **(Simulation: The Tortoise and the Hare)** In this problem, you'll re-create the classic race of the tortoise and the hare. You'll 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 position along the race course. 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. A clock ticks once per second. With each tick of the clock, your application should adjust the position of the animals according to the rules in the table below. Use variables to keep track of the positions of the animals (i.e., position numbers are 1–70). Start each animal at position 1 (the "starting gate"). If an animal slips left before square 1, move it back to square 1.



Animal	Move type	Percentage of the time	Actual move
Tortoise	Fast plod	50%	3 squares 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

Create two functions that generate the percentages in the table for the tortoise and the hare, respectively, by producing a random integer i in the range $1 \leq i \leq 10$. In the function for the tortoise, perform a "fast plod" when $1 \leq i \leq 5$, a "slip" when $6 \leq i \leq 7$ or a "slow plod" when $8 \leq i \leq 10$. Use a similar technique in the function for the hare.

Begin the race by displaying

BANG !!!!!

AND THEY'RE OFF !!!!!

Then, for each tick of the clock (i.e., each iteration of a loop), display a 70-position line showing the letter "T" in the position of the tortoise and the letter "H" in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare, and your application should display "OUCH!!!" at that position. All positions other than the "T", the "H" or the "OUCH!!!" (in case of a tie) should be blank. After each line is displayed, test for whether either animal has reached or passed square 70. If so, display the winner and terminate the simulation. If the tortoise wins, display TORTOISE WINS!!! YAY!!! If the hare wins, display Hare wins. Yuch. If both animals win on the same tick of the clock, you may want to favor the tortoise (the "underdog"), or you may want to display "It's a tie". If neither animal wins, perform the loop again to simulate the next tick of the clock. When you're ready to run your application, assemble a group of fans to watch the race. You'll be amazed at how involved your audience gets!

3- **(Computer-Assisted Instruction)** Computer-assisted instruction (CAI) refers to the use of computers in education. Write a script to help an elementary school student learn multiplication. Create a function that randomly generates and returns a tuple of two positive one-digit integers. Use that function's result in your script to prompt the user with a question, such as How much is 6 times 7? For a correct answer, display the message "Very good!" and ask another multiplication question. For an incorrect answer, display the message "No. Please try again." and let the student try the same question repeatedly until the student finally gets it right.

4- **(Insertion Sort Algorithm)** In computer science, there are several algorithms available to sort the elements of a list such as quicksort, merge, heapsort, and insertion sort. Although insertion sort is not the most efficient sorting algorithm when dealing with large lists, it is very efficient for small data sets and easy to implement. The algorithm splits a list into a sorted and unsorted part. Each value from the unsorted part is sequentially compared to each element in the sorted part and placed in the correct position. The process to sorting the list is:

a) Get a list of unsorted numbers.

b) Compare the first two elements.

c) If they are in descending order, switch them. This is now your sorted sub-list.

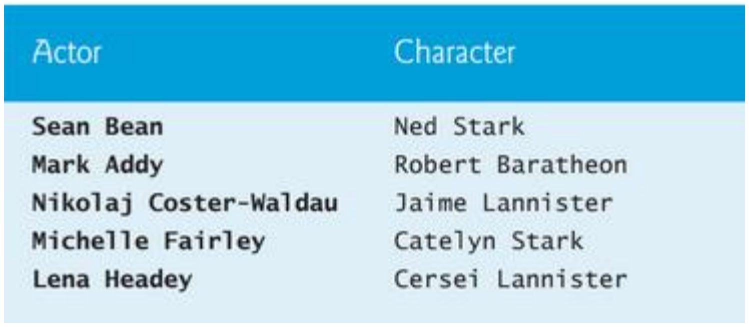
d) Take the third element of the list. Compare, and if necessary, swap it with the second element. Now compare the new second element and swap, if necessary, with the first element.

e) Take the fourth element of the list. Compare and swap with the third element, if necessary.

f) Keep repeating Steps (a) to (e) until every element of the list is in its correct place.

When this process completes, the list of numbers is sorted in ascending order. Write a function insertion sort implementing the described algorithm. Use a list of 10 unique and randomly picked numbers between 0 and 100 to test this function. Display the unsorted and the sorted list to evaluate the validity of your function.

5- **(Guessing Game)** For the TV series “Game of Thrones,” an extensive cast of actors was put to work. Not only is this series one of the most watched of all time but a lot of merchandize has also found its way into the market. Moreover, several games have been launched allowing the player to assume the role of one of the lead characters and experience adventures. Write a script that displays one of the main characters of the series and asks the user which actor played the role. The programmer can choose to either display the actors’ names and let the player chose one or to not display any options at all. The actors and characters are listed in the following table:



Actor	Character
Sean Bean	Ned Stark
Mark Addy	Robert Baratheon
Nikolaj Coster-Waldau	Jaime Lannister
Michelle Fairley	Catelyn Stark
Lena Headey	Cersei Lannister

After each question, the player is informed if their answer was correct. The game is finished when the user has linked five actors to five characters correctly. At the end of the game, the number of correct guesses is displayed on the screen.

6- **(Personnel Files)** Create a dictionary containing information about a company’s staff. Within the dictionary, their personnel numbers are used as dictionary keys. For each staff member, their name, date of birth, and office branch are used as values. Display the dictionary’s contents as a key with an indented list of staff information below it.

7- **(Challenge: Writing the Word Equivalent of a Check Amount)** In check-writing systems, it’s crucial to prevent alteration of check amounts. One common security method requires that the amount be written in numbers and spelled out in words as well. Even if someone can alter the numerical amount of the check, it’s tough to change the amount in words. Create a dictionary that maps numbers to their corresponding word equivalents. Write a script that inputs a numeric check amount that’s less than 1000 and uses the dictionary to write the word equivalent of the amount. For example, the amount 112.43 should be written as ONE HUNDRED TWELVE AND 43/100

8- **(Random Sentences)** Write a script that uses random-number generation to compose sentences. Use four arrays of strings called article, noun, verb and preposition. Create a sentence by selecting a word at random from each array in the following order: article, noun, verb, preposition, article and noun. As each word is picked, concatenate it to the previous words in the sentence. Spaces should separate the words. When the final sentence is output, it should start with a capital letter and end with a period. The script should generate and display 20 sentences.