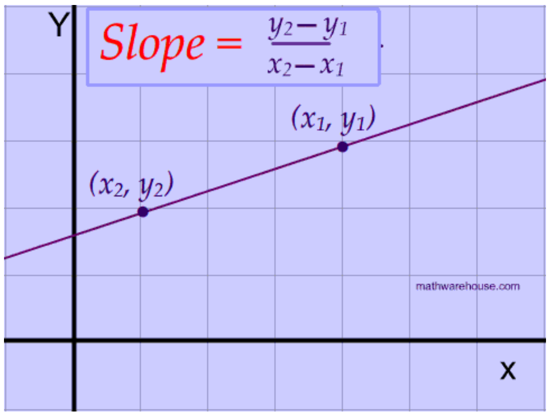
International University
School of Computer Science and Engineering

Fundamentals of Programming

Lab 3
Functions

1. (Fill in the Missing Code) The slope of a line is the number that describes its di- rection and steepness. It is calculated by dividing the change in height by the change in length. Replace the ***s in the **`calc_slope`** function so that it returns the slope of a line drawn in a cartesian coordinate system. The function should receive four integers repre- senting the x and y coordinates of the beginning and end of the line. For example, if you call the function for a line starting at (1,1) and ending at (4,5), the function should return 1.33.



```
def calc_slope(***):
    delta_x = ***
    delta_y = ***
    slope = ***
    return ***
```

2. Write a new **`average`** function to receive **one required argument** and the arbitrary list argument *args, and update its calculation accordingly. Test your function by having it calculate the average of the grades of a student (e.g., 56, 74, 87, 91). When you call **`average`** with no arguments, Python should issue a TypeError indicating "average() missing 1 required positional argument."

```
def average(score1, *args):
```

3. Write a new **`average2`** function to receive the arbitrary list argument **`*args`**, and update its calculation accordingly. Test your function by having it calculate the average of the

grades of a student (e.g., 56, 74, 87, 91). When you call **`average2`** with no arguments, **`average2`** prints out the message ”average() missing 1 required positional argument."

```
def average(*args):
```

4. (Date and Time) Python’s **`datetime`** module contains a **`datetime`** type with a method **`today`** that returns the current date and time as a **`datetime`** object. Write a parameterless **`date_and_time`** function containing the following statement, then call that function to display the current date and time:

   ```
   print(datetime.datetime.today())
   ```

   On our system, the date and time display in the following format:

   ```
   2018-06-08 13:04:19.214180
   ```

5. (Degrees to Radians) Implement a **`radians`** function that returns the radian equivalent of a degree. Use the following formula:

   ```
   rad = degrees * pi/180
   ```

   Use this function to print a chart showing the radian equivalent of all degrees ranging from 1° to 180°. Use two digits of precision for the results. Print the outputs in a neat tabular format.

   ```
   from math import pi
   def radians(degrees):
   ```

6. (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 pos- itive 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.

7. (Computer-Assisted Instruction: Reducing Student Fatigue) Varying the computer’s responses can help hold the student’s attention. Modify the previous exercise so that various comments are displayed for each answer. Possible responses to a correct answer should include 'Very good!', 'Nice work!' and 'Keep up the good work!' Possible re- sponses to an incorrect answer should include 'No. Please try again.', 'Wrong. Try

once more.' and 'No. Keep trying.' Choose a number from 1 to 3, then use that value to select one of the three appropriate responses to each correct or incorrect answer.

8. (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.

9. (Play-the-Multiplication-Game Modification) Modify the previous exercise to count the number of correct multiplications. At the end of the game the total number of correct exercises is displayed on the screen.

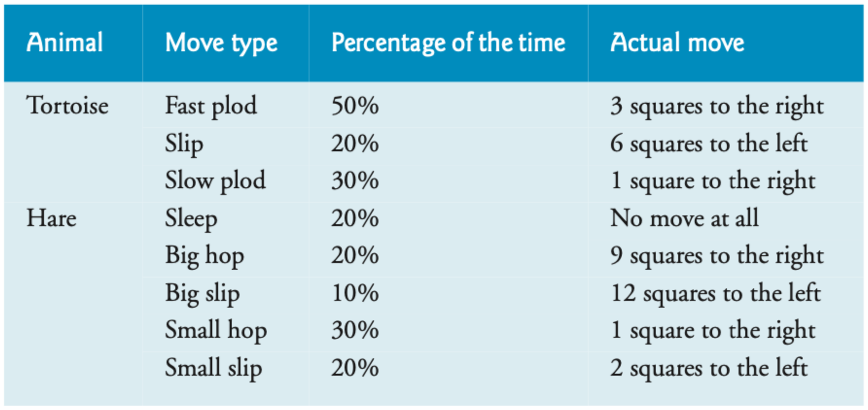
10. **Extra challenge**: You can extend Task 9 further by adding a computer opponent to the game. Write a function that automatically calculates the product of two randomly selected numbers. Add some intelligence to the opponent by implementing smart rules in the function that allow the opponent to make 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.

11. (Computer-Assisted Instruction: Difficulty Levels) Modify the previous exercise to allow the user to enter a difficulty level. At a difficulty level of 1, the program should use only single-digit numbers in the problems and at a difficulty level of 2, numbers as large as two digits.

12. (Computer-Assisted Instruction: Varying the Types of Problems) Modify the previous exercise to allow the user to pick a type of arithmetic problem to study—1 means addition problems only, 2 means subtraction problems only, 3 means multiplication problems only, 4 means division problems only (avoid dividing by 0) and 5 means a random mixture of all these types.

13. **(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.



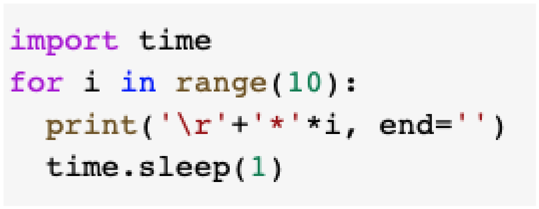
Animal	Move type	Percentage of the time	Actual move
Tortoise	Fast plod	50%	3 squares to the right
	Slip	20%	6 squares to the left
	Slow plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big hop	20%	9 squares to the right
	Big slip	10%	12 squares to the left
	Small hop	30%	1 square to the 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 ≤ i ≤ 10. In the function 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 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, dis- play 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!

Hint: in order to overwriting output on the same line, try the code below.



```
import time
for i in range(10):
  print('\r'+'*'*i, end='')
  time.sleep(1)
```