# Simulations – Assignment03

Loops, conditional statements and functions can all be used together to create simulations. Simulations are representations of events in the real world. But, before we can create a meaningful simulation of any kind, we need to first become accustomed to using all those language elements together in the same assignment.

In this assignment, you will simulate the roll of dice and then use those dice values to simulate how many pizzas Cheesy Pete’s Pizza Palace will sell over one more days.

|  |
| --- |
| **Watch out!** We’ll need to use a little jQuery along without JavaScript to make it all work. Be sure to read the gray boxes like this one. |

|  |
| --- |
| ***Advisory*** – Students are expected to complete this assignment, on their own, using the contents of PowerPoint presentations and ICAs used in Weeks 1, 2, 3, 4 and 5 this semester. There is no reason to use AI to solve this. Give it a try, all by yourself. You can do it. (And, please take a second to review the “Academic Integrity” and “Style Guide” sections of the syllabus.) |

## Instructions

1. Download assignment03.zip and put your assignment03 folder into your MIS2402 workspace.
2. Start by editing simulation1.html – this page will simulate the roll of a single die.

## Scenario 1 – Roll a single, six-sided die

|  |  |
| --- | --- |
| There are many kinds of dice. The most common die has six sides.  A roll of a fair six-sided die should return a uniformly distributed random integer between 1 and 6. This first scenario is really very simple. ***There is no loop in this scenario.*** |  |

1. In simulation1.html observe that a function called getRandomInt() has been defined for you. This function will return a uniformly distributed random number between 1 and x. So getRandomInt(3) might return 1, 2, or 3.
2. End the existing click event handler for btn\_1. You will use this to call the getRandomInt function, put the result of that function into variable ( you can call the variable answer if that is helpful). Then, put the answer into the inner html of the existing tag textDisplayed1 with a command similar to the following:   
     
   $("#textDisplayed1").html(answer);

|  |
| --- |
| **That’s jQuery.** The statement means, find the HTML tag with the id of textDisplayed1 and put the value of answer into the inner html of that tag. |

1. Test your work. You should see random numbers between 1 and 6 when you click the “Calculate” button. (That was easy, right?)

## Scenario 2 – Simulate the roll two dice at the same time

|  |  |
| --- | --- |
| When you play a board game, you often roll two dice at the same time. In other games, you might roll three or more dice. In this scenario we will extend the work we did in scenario one to simulate a roll of ***two*** dice.  Think carefully here. The roll of two dice does not produce uniformly distributed random numbers between 1 and 12. Neither does it produce uniformly distributed random numbers between 2 and 12. The roll will be the sum of two independent dice rolls. |  |

Seven is the most likely number when you roll two six-sided dice. This is common knowledge. So, you need to stop here and think how exactly your code will simulate this.

1. Edit simulation2.html – this page will simulate the roll of 2 dice. Complete the roll2Dice()function. This function should ***call*** the existing getRandomInt() function and return the sum of the two dice.
2. Now, complete the statCheck() function. The purpose of the function is to determine the average outcome of roll2Dice()function. This will require a loop. Use a loop to get the sum of 1000 rolls of 2 dice. Then divide by 1000. This will be the average value of a 1000 rolls. Return this value.
3. **Test your work.** Because your result depends on random values, there’s no guarantee that you will always get exactly the same result. But, if you have done everything right, your answer should be *approximately* seven.

Here are some sample outcomes when you click the “Stat Check” button.

A screenshot of a game

Description automatically generated A screenshot of a game

Description automatically generated A screenshot of a game

Description automatically generated

CONTINUED

## Scenario 3 – Data Validation / Error Trapping

1. Edit simulation3.html. In this file you will need to copy/paste in the isNaturalNumber() function that your created earlier in the semester. You should also bring in (that is copy/paste) the roll2Dice() function you created in scenario 2.
2. Take note of the function, renderForecast() it is already begun for you. Your job now is to test if the variable named n is not or is not a natural number. If it’s not a natural number, it should place the text “Bad data. Try again.” into the textDisplayed1 html tag.   
     
   The jQuery command to do that looks like this:

$("#textDisplayed1").html("Bad data. Try again.");

1. If the value of n is good, then the function should execute a loop that iterates n times. With each iteration, you roll 2 dice and total the values. When the loop is done, calculate the average and put it into the inner html of the existing tag textDisplayed1 with a command similar to the following:

$("#textDisplayed1").html(answer); //more jQuery here

1. **Test your work.** Your error trapping should work. As before, your output will vary somewhat due to randomness. But, the larger the value of n, the closer your average will get to seven.

**Examples:**

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer game

AI-generated content may be incorrect. A screenshot of a computer game

AI-generated content may be incorrect.

## Scenario 4 – Pizza Demand Forecast

1. Open simulation4.html. Read the instructions found in the JavaScript comments. There are step-by-step instructions there and you should read and follow them.

Use your knowledge of conditional statements, loops and functions to complete the code necessary to generate a forecast for the number of pizzas sold.  
  
Pseudo code follows on the next page for the trickiest part… the “renderForecast” function.  
  
  
  
  
Pseudo code for the “renderForecast” function in step 13

* if n is not a natural number, that's bad. Report "Bad data. Try again." in textDisplayed 1 and stop.
* if n is good, then get to work!
* create a variable to store the demand for a single day. It should have an initial value of zero.
* create a second variable to store the total demand. It should have an initial value of zero.
* write a loop that goes from 1 to n, inclusive. These are the days of your simulation: day 1, day 2, etc.
* Inside the loop
  + calculate the demand for a single day by calling the getDemand function and assign it to the demand variable you created earlier
  + add the single day demand to the existing total demand
  + call getPizzaIcons to get the string of html you want to display.
  + use the jQuery append method to add that string on to the contents of textDiplayed1
  + For example: $("#textDisplayed1").append("EXAMPLE");
* When the loop is done. Calculate the average demand by dividing the total demand by n.
* Round your answer to two decimal places.
* Put the answer into textDisplayed2 using jQuery. $("#textDisplayed2").html("EXAMPLE");

|  |
| --- |
| **There’s a function, already written for you, called getPizzaIcons**. You should use it. Don’t change it. It is set up to return 1 pizza icon for every 40 pizzas. It also has some tricky logic to deal with values of n that are not cleanly divisible by 40. It is already written. For you. To use. |

1. Test your work. Your forecast will vary randomly. Don’t forget to test your error trapping.

See the next page for some examples:

A screenshot of a computer game

AI-generated content may be incorrect.

A screenshot of a video game

AI-generated content may be incorrect.

A screenshot of a cell phone

AI-generated content may be incorrect.

This is the average pizza demand over these 7 days.

HINT – we are not using an HTML ordered list here. In this case, the numbering (1,2,3, etc) is coming from the getPizzaIcons function.

## When you are done…

1. Be sure to upload both all four html files to assignment 3 on canvas.

How will this assignment be graded?

Scenario 1 work is worth 25 points. Scenario 1 needs to run. If it does not run at all you lose all 25 points.

Scenario 2 work is worth 25 points. Scenario 2 needs to run. If it does not run at all you lose all 25 points.

Scenario 3 work is worth 25 points. Scenario 3 needs to run. If it does not run at all you lose all 25 points.

Scenario 4 work is worth 25 points. Scenario 4 needs to run. If it does not run at all you lose all 25 points.

Assuming your work runs – any other point deductions will be assigned in 5-point increments.

Things you could lose points for:

* Missing / incomplete error trapping in scenarios 3 and 4.
* Solving the problem without loops / or with unexpected language elements not taught in class
* Renaming functions
* Miscalculations
* Any bug / error that causes the program return incorrect results
* Neglecting to round your results in simulation