Week 2

Task 1.1: Loops

The `SimpleLoop.java` program shown below contains a loop which calculates n^n for $0 < n < 9$. It does this by for-looping over values from n = 0 up to (but excluding) n = `maxn` = 10. Next, the program generates a random number between 0 and 99, and asks the user to guess the number. I modified the code so that it tells the user whether their guess in correct or incorrect, and, if incorrect, tells them to make another guess higher or lower than their last. I did this using `if`, `else if` and `else` statements. The program also prints the number of tries the user took to correctly guess the number. An example of the output is shown beneath the code:

```
1.  package week2;
2.
3.  import java.util.Random;
4.  import java.util.Scanner;
5.
6.  class Week2 {
7.
8.      static Scanner keyboard = new Scanner(System.in);
9.
10.     public static void main(String[] args) {
11.
12.         // example 1 , "for" loop.
13.         int maxn = 10;
14.         for (int n = 0; n < maxn; n++) { // n++ means add 1 to current value of n
15.             System.out.println("When n = " + n + " then n^n = " + Math.pow(n,n) );

16.         }
17.
18.         // example 2 , "while" loop
19.         // guessing game.. try to guess the number I'm thinking of
20.         Random randomGenerator = new Random();
21.         int answer = randomGenerator.nextInt(100); // generates an int between 0 an
    d 99
22.         System.out.println("Hello.. try to guess the integer in the range 0 to 99 I
    'm thinking about");
23.         System.out.println("Type in your first guess");
24.
25.         int yourGuess = -1;
26.         int count = 0;
27.
28.         while (yourGuess != answer) { // != means NOT EQUAL
29.             yourGuess = keyboard.nextInt();
30.
31.             if (yourGuess < answer) {
32.                 System.out.println("No, your guess = " + yourGuess + " is not corre
    ct, try higher");
33.             }
34.             else if (yourGuess > answer) {
35.                 System.out.println("No, your guess = " + yourGuess + " is not corre
    ct, try lower");
36.                 //'else if' runs in the first condition wasn't met
37.             }
38.             else {
39.                 System.out.println("Correct!");
40.                 //runs in none of the conditions are met, the guess must be correct
    !
41.             }
42.         count = count +1;
43.         }
```

```
44.         System.out.println("Yes, "+ answer + " is correct. You guessed it in " + co
    unt + " tries!");
45.     }
46. }
```

Output:

```
run:
When n = 0 then n^n = 1.0
When n = 1 then n^n = 1.0
When n = 2 then n^n = 4.0
When n = 3 then n^n = 27.0
When n = 4 then n^n = 256.0
When n = 5 then n^n = 3125.0
When n = 6 then n^n = 46656.0
When n = 7 then n^n = 823543.0
When n = 8 then n^n = 1.6777216E7
When n = 9 then n^n = 3.87420489E8
Hello.. try to guess the integer in the range 0 to 99 I'm thinking about
Type in your first guess
50
No, your guess = 50 is not correct, try lower
30
No, your guess = 30 is not correct, try higher
40
No, your guess = 40 is not correct, try lower
35
Correct!
Yes, 35 is correct. You guessed it in 4 tries!
BUILD SUCCESSFUL (total time: 13 seconds)
```

Task 1.2: Arrays and Loops

The following program stores a certain number of integers in an array. The user is asked how many numbers they wish to store in the array, and the array is initialised with the required length. The user is then asked to type each of the numbers in. The program uses a for loop to add each element to the array. I added code which calculates sum of the numbers in the array by for-looping over the length of the array and adding each number to the variable `sum`.

If the user has entered an array length greater than 0, the array and the arithmetic mean (the sum divided by the length of the array) is printed. If the length is 0, the user is told, 'There are no numbers in the array!' The code was tested in two cases:

- Case 1: 5 numbers, 1, 4, 7, 3, 4
- Case 2: zero (no) numbers

The output in both cases is shown below the code.

```
1.  package task12;
2.
3.  import java.util.Scanner;
4.
5.  public class Task12 {
6.
7.      static Scanner keyboard = new Scanner(System.in);
```

```
8.
9.      public static void main (String [] args )
10.     {
11.         // Program to store integer numbers in an array
12.         System.out.println("How many numbers do you want to store?");
13.         int numberToStore = keyboard.nextInt();
14.         double [] myStore = new double [numberToStore];
15.
16.         // create an array called 'myStore' in memory with this number of elements

17.         // remember: the array index start at ZERO, not 1.
18.         for (int n = 0; n < numberToStore; n++) {
19.             System.out.println( "Please type in the " + (n+1) + " number     " );
20.             myStore[n] = keyboard.nextInt();
21.         }
22.
23.         double sum = 0.; //start the sum at 0
24.         for (int i = 0; i < numberToStore; i++) { //going up to the length of the a
    rray
25.             sum += myStore[i]; //add the numbers to the current total
26.         }
27.
28.         System.out.println("\n\nAll finished... please check this list:");
29.         if (numberToStore == 0){
30.             System.out.println("\nThere are no numbers in the array!");
31.         }
32.         else {
33.
34.             for (int n = 0; n < numberToStore; n++) {
35.                 System.out.println( n + "  location,  value stored = " + myStore[n]
    );
36.             }
37.             double average = sum / numberToStore; //calculating the average
38.             System.out.println("\nThe average of these numbers = " + average);
39.         }
40.     }
41. }
```

Case 1 Output:

```
run:
How many numbers do you want to store?
5
Please type in the 1 number
1
Please type in the 2 number
4
Please type in the 3 number
7
Please type in the 4 number
3
Please type in the 5 number
4


All finished... please check this list:
0  location,  value stored = 1.0
1  location,  value stored = 4.0
2  location,  value stored = 7.0
3  location,  value stored = 3.0
4  location,  value stored = 4.0
```

```
The average of these numbers = 3.8
BUILD SUCCESSFUL (total time: 10 seconds)
```

Case 2 Output:

```
run:
How many numbers do you want to store?
0


All finished... please check this list:

There are no numbers in the array!
BUILD SUCCESSFUL (total time: 2 seconds)
```

Task 2

The program `GenerateHistogram.java` uses the class `Histogram` to generate a histogram using random numbers between 0 and 1. The data is then written to a csv, and was plotted in Excel. The code is shown below and described in Tasks 2.1, 2.2 and 2.3.

GenerateHistogram.java

```java
package generatehistograms;

import java.util.Scanner;
import java.util.Random;

class GenerateHistograms
{
    static Random randGen = new Random();
    static Scanner keyboard = new Scanner(System.in);

    private static double addTwelve()
    {
        // adds 12 uniformly-distributed random numbers
        double sum = 0.;
        for (int n = 0 ; n < 12; n++) {
            sum = sum + randGen.nextDouble();
        }
        return (sum);
    }

    public static void main (String [] args)
    {
        // create an instance of the Class Histogram: 50 bins from 0.0 to 1.0
        // this is for the flat distribution
        Histogram hist = new Histogram(50, 0.2, 0.9, "Uniform");

        System.out.println( "Input the number of random numbers to generate");
        int trials = keyboard.nextInt();
        for (int i = 0; i < trials; i++) {
            double value = randGen.nextDouble();
            hist.fill(value);
        }

        hist.print();
        hist.writeToDisk("test.csv");
```

```
36.
37.         Histogram hist2 = new Histogram(50, 3, 9, "Uniform");
38.         int trials2 = 1000;
39.         for (int i = 0; i < trials2; i++) {
40.             double value2 = addTwelve();
41.             hist2.fill(value2);
42.         }
43.
44.         hist2.print(); //printing the histogram used in Task 2.3
45.         hist2.writeToDisk("His2Test.csv"); //saving the data to a csv
46.     }
47. }
```

Histogram.java

```
1.  package generatehistograms;
2.
3.  // this import is needed for the file input/output functionality
4.  import java.io.*;
5.  import java.lang.Math;
6.
7.  class Histogram
8.  {
9.      private double binlow, binhigh;
10.     private double binwidth;
11.     private int nbins;
12.     private double[] binCentre;
13.     private String histname;
14.     private int underCount = 0; //the number of values below the bounds of the hist
    ogram
15.     private int overCount = 0; //the number of values above the bounds
16.     private int insideCount = 0; //the number of values within the bounds
17.
18.     // double array to store the actual histogram data
19.     private double[] sumWeights;
20.
21.     // constructor for the class Histogram
22.     public Histogram(int numberOfBins, double start, double end, String name)
23.     {
24.         // store the parameters and setup the histogram
25.         // note that parameters need to have different names than class variables
26.         nbins = numberOfBins;
27.         binlow = start;
28.         binhigh = end;
29.         histname = name;
30.
31.         binwidth = (binhigh - binlow) / (double) nbins;
32.         sumWeights = new double[nbins];
33.
34.         // calculate and save the x coordinate of the centre of each bin
35.         binCentre = new double[nbins];
36.         for (int i = 0; i < nbins; i++) {
37.             binCentre[i] = binlow + (i+0.5)*binwidth;
38.         }
39.     }
40.
41.     public int getNbins()
42.     {
43.         return nbins;
44.     }
45.
46.     public void fill(double value)
47.     {
48.         if (value < binlow){
49.             underCount += 1;
```

```
50.         }
51.         else if (value > binhigh) {
52.             overCount += 1;
53.         }
54.         else{
55.             //if it is within the bounds of the histogram then it finds the appropr
    iate bin and then adds 1 to that bin's total (puts the value in the bin)
56.             // find correct bin and add 1.
57.             int ibin = (int) ( (value - binlow)/binwidth);
58.             sumWeights[ibin] = sumWeights[ibin] + 1.0;
59.             insideCount += 1;
60.         }
61.     }
62.
63.     public double getContent(int nbin)
64.     {
65.         // returns the contents on bin 'nbin' to the user
66.         return sumWeights[nbin];
67.     }
68.
69.     public double calcError(int nbin)
70.     {
71.         return Math.sqrt(sumWeights[nbin]);
72.     }
73.
74.     public void print()
75.     {
76.         System.out.println("Histogram " + histname);
77.         for (int bin = 0; bin < getNbins(); bin++) {
78.             System.out.println("Bin " + bin + " = " +getContent(bin) + "+/- " + cal
    cError(bin));
79.         }
80.         System.out.println("Underflow = " + underCount + " and overflow = " + overC
    ount);
81.         System.out.println("The histogram was filled " + insideCount + " times");
82.     }
83.
84.     public void writeToDisk(String filename)
85.     {
86.         // this sends the output to a file with name "filename"
87.         // the block with try { ... } catch (IOException e) { ... } is needed to ha
    ndle the case,
88.         // where opening the file fails, e.g. disk is full or similar
89.         PrintWriter outputFile;
90.         try {
91.             outputFile = new PrintWriter(filename);
92.         } catch (IOException e) {
93.             System.err.println("Failed to open file " + filename + ". Histogram dat
    a was not saved.");
94.             return;
95.         }
96.
97.         // Write the file as a comma seperated file (.csv) so it can be read it int
    o EXCEL
98.         // first some general information about the histogram
99.         outputFile.println("histname, " + histname);
100.                outputFile.println("binlow, " + binlow);
101.                outputFile.println("binwidth, " + binwidth);
102.                outputFile.println("nbins, " + nbins);
103.
104.                // now make a loop to write the contents of each bin to disk, one nu
    mber at a time
105.                // together with the x-coordinate of the centre of each bin.
106.                for (int n = 0; n < nbins; n++) {
107.                    // comma separated values
```

```
108.                     outputFile.println(n + "," + binCentre[n] + "," + getContent(n)
    + "," + calcError(n));
109.                 }
110.                 outputFile.close(); // close the output file
111.             }
112.         }
```

Task 2.1

In this task, I extended the class `Histogram` by a method which returns the error for a certain bin (the square root of the number of counts in that bin). I did this by adding the following method to `Histogram.java`:

```
1.  public double calcError(int nbin)
2.  {
3.      return Math.sqrt(sumWeights[nbin]);
4.  }
```

The method takes an integer input, `nbin` (the index, n, of the bin), finds the number of counts in the n^{th} bin of the `sumWeights` array, and finds the square root of the counts. 100 random numbers were generated in `GenerateHistogram.java`, the contents of each bin and the error were printed to the screen and saved in a csv file. The data were plotted in *Figure 1*. An example of the print out to screen is shown in Task 2.2.

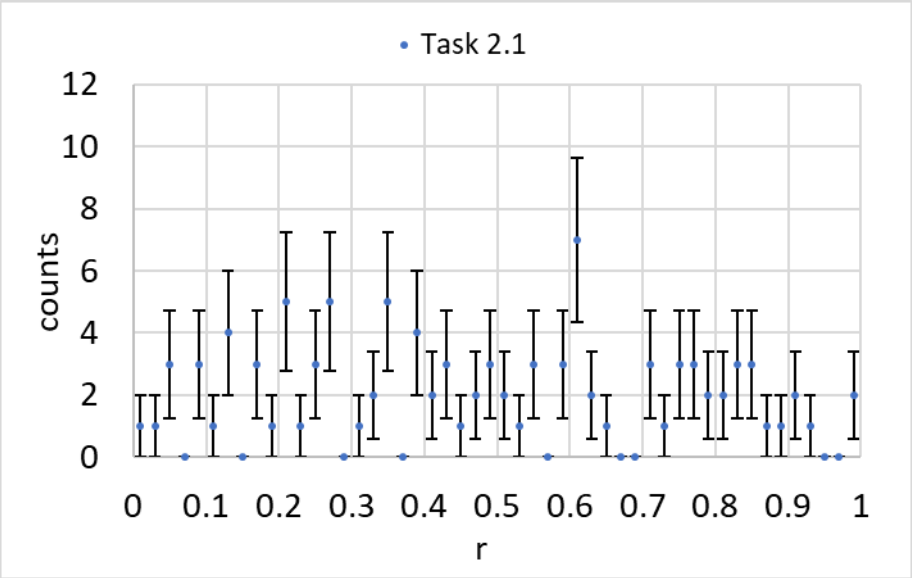

Figure 1: Counts vs r plotted using 50 bins and 0 < r < 1.

Task 2.2

I modified the method `fill()` in `Histogram.java` so that it would count the number of underflows, overflows, and the number of times the histogram was filled, then print these values. I did this using `if`, `else if` and `else` statements. If r (`value`) was below the range of the histogram, 1 was added to the number of underflows. If r was above the range, 1 was added to the

number of overflows. If r was within the bounds, 1 was added to the number of times the histogram was filled:

```
1.      public void fill(double value)
2.      {
3.          if (value < binlow){  //binlow is the minimum bound of the histogram
4.              underCount += 1;  // undercount is the number of underflows
5.          }
6.          else if (value > binhigh) {  //binhigh is the maximum bound
7.              overCount += 1;
8.          }
9.          else{
10.             //if it is within the bounds of the histogram then it finds the appropr
    iate bin and then adds 1 to that bin's total (puts the value in the bin)
11.             // find correct bin and add 1.
12.             int ibin = (int) ( (value - binlow)/binwidth);
13.             sumWeights[ibin] = sumWeights[ibin] + 1.0;
14.             insideCount += 1;
15.         }
16.     }
```

Using the bounds 0 < r < 1 in `GenerateHistograms.java` gave 0 underflows and overflows. This was expected, because the random numbers being generated were between 0 and 1. To test the program, I reduced the bounds to cover the range 0.2 < r < 0.9. This had 22 underflows and 6 overflows; a total of 28 counts outside the bounds. The bounds were reduced by 30%, so it was expected that 30% of the counts should be outside the bounds. 28/100 is close to 30%, as expected. The histogram is shown in *Figure 2* and the output of the code is shown beneath it.

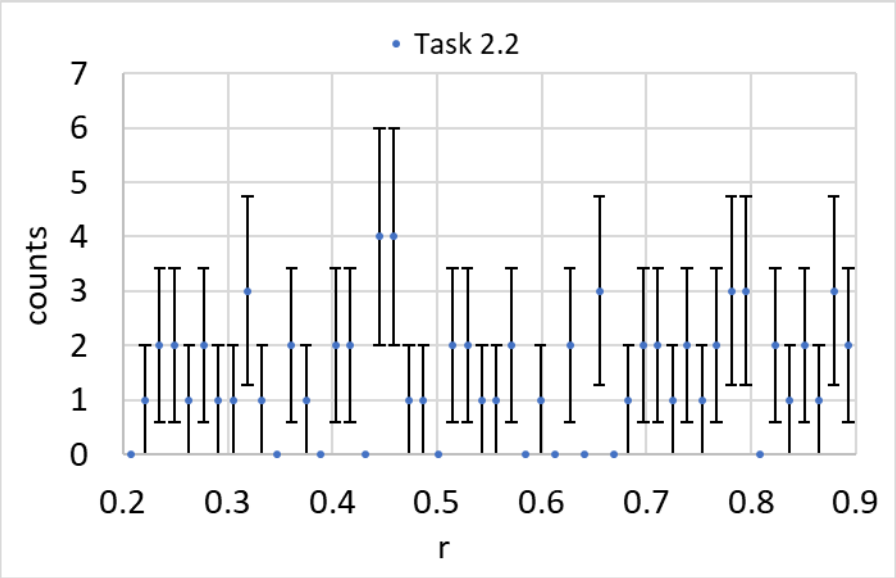

Figure 2: The counts vs r histogram generated using 100 random numbers when the bounds were reduced to 0.2 < r < 0.9.

Output:

```
run:
Input the number of random numbers to generate
100
Histogram Uniform
Bin 0 = 2.0+/- 1.4142135623730951
Bin 1 = 0.0+/- 0.0
```

```
Bin 2 = 3.0+/- 1.7320508075688772
Bin 3 = 4.0+/- 2.0
Bin 4 = 4.0+/- 2.0
Bin 5 = 0.0+/- 0.0
Bin 6 = 1.0+/- 1.0
Bin 7 = 3.0+/- 1.7320508075688772
Bin 8 = 1.0+/- 1.0
Bin 9 = 1.0+/- 1.0
Bin 10 = 2.0+/- 1.4142135623730951
Bin 11 = 2.0+/- 1.4142135623730951
Bin 12 = 0.0+/- 0.0
Bin 13 = 1.0+/- 1.0
Bin 14 = 2.0+/- 1.4142135623730951
Bin 15 = 2.0+/- 1.4142135623730951
Bin 16 = 0.0+/- 0.0
Bin 17 = 0.0+/- 0.0
Bin 18 = 1.0+/- 1.0
Bin 19 = 0.0+/- 0.0
Bin 20 = 3.0+/- 1.7320508075688772
Bin 21 = 0.0+/- 0.0
Bin 22 = 1.0+/- 1.0
Bin 23 = 0.0+/- 0.0
Bin 24 = 0.0+/- 0.0
Bin 25 = 2.0+/- 1.4142135623730951
Bin 26 = 0.0+/- 0.0
Bin 27 = 1.0+/- 1.0
Bin 28 = 1.0+/- 1.0
Bin 29 = 2.0+/- 1.4142135623730951
Bin 30 = 2.0+/- 1.4142135623730951
Bin 31 = 1.0+/- 1.0
Bin 32 = 0.0+/- 0.0
Bin 33 = 2.0+/- 1.4142135623730951
Bin 34 = 1.0+/- 1.0
Bin 35 = 1.0+/- 1.0
Bin 36 = 4.0+/- 2.0
Bin 37 = 1.0+/- 1.0
Bin 38 = 1.0+/- 1.0
Bin 39 = 1.0+/- 1.0
Bin 40 = 2.0+/- 1.4142135623730951
Bin 41 = 2.0+/- 1.4142135623730951
Bin 42 = 3.0+/- 1.7320508075688772
Bin 43 = 6.0+/- 2.449489742783178
Bin 44 = 0.0+/- 0.0
Bin 45 = 1.0+/- 1.0
Bin 46 = 1.0+/- 1.0
Bin 47 = 1.0+/- 1.0
Bin 48 = 1.0+/- 1.0
Bin 49 = 2.0+/- 1.4142135623730951
Underflow = 22 and overflow = 6
The histogram was filled 72 times
BUILD SUCCESSFUL (total time: 2 seconds)
```

Task 2.3

For the final task, I prepared a second histogram to record the distribution of random numbers generated using the routine `addTwelve()`, defined at the start of `GenerateHistogram.java`. `addTwelve()`, sums 12 random numbers between 0 and 1:

```
1.      private static double addTwelve()
2.      {
3.          // adds 12 uniformly-distributed random numbers
4.          double sum = 0.;
5.          for (int n = 0 ; n < 12; n++) {
6.              sum = sum + randGen.nextDouble();
7.          }
8.          return (sum);
9.      }
```

I added the following code to `main` in `GenerateHistogram.java` to generate the second histogram from 1000 random numbers and to store the data:

```
1.          Histogram hist2 = new Histogram(50, 3, 9, "Uniform");  //defining the
    second histogram, hist2, and using the Histogram Class
2.          int trials2 = 1000; // the number of random numbers to generate
3.          for (int i = 0; i < trials2; i++) {
4.              double value2 = addTwelve();  //the 1000 numbers were generated using
    the routine addTwelve()
5.              hist2.fill(value2);
6.          }
7.
8.          hist2.print(); //printing the histogram used in Task 2.3
9.          hist2.writeToDisk("His2Test.csv"); //saving the data to a csv
```

I ran the program a few times in order to decide what bounds to use. Bins below 3 and above 9 were empty in most runs, so I reduced the bounds to $3 < r < 9$. Running the program after the bounds were adjusted gave 1 underflow and 2 overflows, a very small fraction of the total (1000). I used 50 bins to give the histogram sufficient definition. The results are plotted in *Figure 3*.

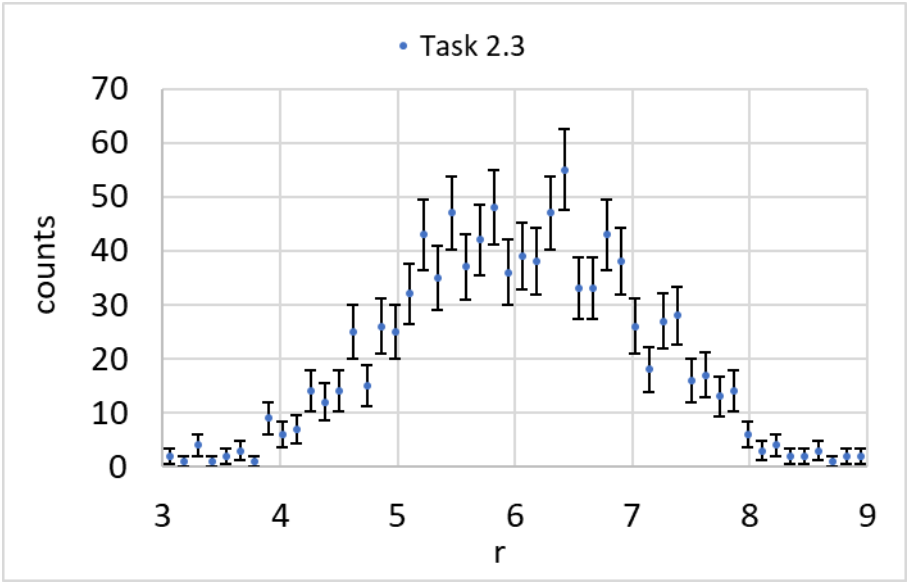

Figure 3: A histogram showing counts vs r for the sum of 12 random numbers, generated 1000 times. The bounds were constricted to 3 < r < 9.

Figure 3 exhibits a normal distribution due to Central Limit Theorem. Central Limit Theorem states that when independent random variables are added, their normalised sum tends towards a normal distribution [1]. The counts have not been normalised in *Figure 3*, but the bell curve shape indicates that summing 12 random numbers a large number of times gives a normal distribution. The output of the code is shown below.

Output:

```
Histogram Uniform
Bin 0 = 2.0+/- 1.4142135623730951
Bin 1 = 1.0+/- 1.0
Bin 2 = 4.0+/- 2.0
Bin 3 = 1.0+/- 1.0
Bin 4 = 2.0+/- 1.4142135623730951
Bin 5 = 3.0+/- 1.7320508075688772
Bin 6 = 1.0+/- 1.0
Bin 7 = 9.0+/- 3.0
Bin 8 = 6.0+/- 2.449489742783178
Bin 9 = 7.0+/- 2.6457513110645907
Bin 10 = 14.0+/- 3.7416573867739413
Bin 11 = 12.0+/- 3.4641016151377544
Bin 12 = 14.0+/- 3.7416573867739413
Bin 13 = 25.0+/- 5.0
Bin 14 = 15.0+/- 3.872983346207417
Bin 15 = 26.0+/- 5.0990195135927845
Bin 16 = 25.0+/- 5.0
Bin 17 = 32.0+/- 5.656854249492381
Bin 18 = 43.0+/- 6.557438524302
Bin 19 = 35.0+/- 5.916079783099616
Bin 20 = 47.0+/- 6.855654600401044
Bin 21 = 37.0+/- 6.082762530298219
Bin 22 = 42.0+/- 6.48074069840786
Bin 23 = 48.0+/- 6.928203230275509
Bin 24 = 36.0+/- 6.0
Bin 25 = 39.0+/- 6.244997998398398
Bin 26 = 38.0+/- 6.164414002968976
Bin 27 = 47.0+/- 6.855654600401044
Bin 28 = 55.0+/- 7.416198487095663
Bin 29 = 33.0+/- 5.744562646538029
Bin 30 = 33.0+/- 5.744562646538029
Bin 31 = 43.0+/- 6.557438524302
Bin 32 = 38.0+/- 6.164414002968976
Bin 33 = 26.0+/- 5.0990195135927845
Bin 34 = 18.0+/- 4.242640687119285
Bin 35 = 27.0+/- 5.196152422706632
Bin 36 = 28.0+/- 5.291502622129181
Bin 37 = 16.0+/- 4.0
Bin 38 = 17.0+/- 4.123105625617661
Bin 39 = 13.0+/- 3.605551275463989
Bin 40 = 14.0+/- 3.7416573867739413
Bin 41 = 6.0+/- 2.449489742783178
Bin 42 = 3.0+/- 1.7320508075688772
```

```
Bin 43 = 4.0+/- 2.0
Bin 44 = 2.0+/- 1.4142135623730951
Bin 45 = 2.0+/- 1.4142135623730951
Bin 46 = 3.0+/- 1.7320508075688772
Bin 47 = 1.0+/- 1.0
Bin 48 = 2.0+/- 1.4142135623730951
Bin 49 = 2.0+/- 1.4142135623730951
Underflow = 1 and overflow = 2
The histogram was filled 997 times
BUILD SUCCESSFUL (total time: 2 seconds)
```

Works Cited

[1] MathWorld, "Central Limit Theorem," Wolfram, [Online]. Available: http://mathworld.wolfram.com/CentralLimitTheorem.html. [Accessed 05 02 2020].