(http://csmatters.org) 4 - 5

0b100 - 0b101

Data Collection, Analysis, and Simulation

Unit 4. Data Acquisition

Revision Date: Sep 22, 2019
Duration: 2 50-minute sessions

Lesson Summary

Pre-lesson Preparation

Your students will need computers for this lesson. If you would like to show students a working dartboard simulation (http://mw2.concord.org/public/student/game/dartboard.html) (with a circular dartboard), check that your browser can run a Java plug-in. Be sure to update, activate, and disable the plug-in as needed for security purposes.

Summary

In this lesson, students will explore basic data analysis concepts in Python, learn about code extensibility, create a simple simulation from scratch, and reuse their code to make a more elaborate simulation.

Outcomes

- Students will create a mathematical simulation and understand how programming can be used to model real-world processes.
- Students will understand extensibility and code reuse by developing a simulation and modifying it to solve a more complex task.
- Students will be able to reason about and solve a problem by programming a solution from scratch.
- Students will collect and analyze data.
- Students will understand that analog data can be closely approximated digitally using a sampling technique, which means measuring values of the analog signal at regular intervals called samples and that the samples are measured to figure out the exact bits required to store each sample.

Overview

Session 1:

1. Getting Started (5 min)

2. Guided Activity (45 min)
3. Optional Homework

Session 2:

1. Getting Started (5 min)
2. Guided Activity (45 min)
3. Homework

Part of this lesson was adapted from http://www.nzmaths.co.nz/resource/dartboards (http://www.nzmaths.co.nz/resource/dartboards) and http://www.nzmaths.co.nz/resource/more-dartboards (http://www.nzmaths.co.nz/resource/more-dartboards).

Learning Objectives

CSP Objectives

- *EU CRD-2 - Developers create and innovate using an iterative design process that is user-focused, that incorporates implementation/feedback cycles, and that leaves ample room for experimentation and risk-taking.*
  - LO CRD-2.E - Develop a program using a development process.
  - LO CRD-2.F - Design a program and its user interface.
- *EU DAT-2 - Programs can be used to process data, which allows users to discover information and create new knowledge.*
  - LO DAT-2.E - Explain how programs can be used to gain insight and knowledge from data.
- *EU AAP-2 - The way statements are sequenced and combined in a program determines the computed result. Programs incorporate iteration and selection constructs to represent repetition and make decisions to handle varied input values.*
  - LO AAP-2.B - Represent a step-by-step algorithmic process using sequential code statements.
  - LO AAP-2.C - Evaluate expressions that use arithmetic operators.
  - LO AAP-2.G - Express an algorithm that uses selection without using a programming language.
  - LO AAP-2.H - For selection: a. Write conditional statements. b. Determine the result of conditional statements.
  - LO AAP-2.J - Express an algorithm that uses iteration without using a programming language.
  - LO AAP-2.K - For iteration: a. Write iteration statements. b. Determine the result or side-effect of iteration statements.
- *EU AAP-3 - Programmers break down problems into smaller and more manageable pieces. By creating procedures and leveraging parameters, programmers generalize processes that can be reused. Procedures allow programmers to draw upon existing code that has already been tested, allowing them to write programs more quickly and with more confidence.*

- LO AAP-3.A - For procedure calls: a. Write statements to call procedures. b. Determine the result or effect of a procedure call.
- LO AAP-3.F - For simulations: a. Explain how computers can be used to represent real-world phenomena or outcomes. b. Compare simulations with real-world contexts.

Key Concepts

The development of a program from scratch to solve a specific problem is presented to students by creating a simulation that lets them see how software can model a real-world process. Additionally, the concepts of extensibility and code reuse are shown through hands-on programming experience.

Some real-world problems have digital input data, many have analog (sound, light, pictures, motion, temperature, etc.) which needs to be considered when determining how to gather input and what to use as input.

Essential Questions

- How are algorithms implemented and executed on computers and computational devices?
- How are programs used for creative expression, to satisfy personal curiosity or to create new knowledge?
- How do computer programs implement algorithms?

Teacher Resources

Student computer usage for this lesson is: **required**

The Lesson Resources folder contains an example program showing how to use Python's random function to simulate tossing a coin, as well as a Dartboard.py solution. There is also a comparison between this python code, and some pseudocode that one may write before coding Dartboard.py, titled "Pseudocode vs Python."

An alternative lesson outline using Runestone and PyCharm to code a simulation and use it to develop, refine and test hypotheses in is the lesson folder. The lesson is in a file named "Monte Carlo Simulation to Calculate Pi.docx".

Lesson Plan

Session 1

Getting Started (5 min)

Think-Pair-Share: Writing programs "from scratch"

- Ask your students to think about what it means to write a program "from scratch," how and why someone would do so.
- Have your students pair off to share and list their ideas.
- Collect and discuss their responses. Some topics to note could include: the importance of programming to solve a specific problem, the challenge of starting a program without copying someone else's code, and the value of learning to implement algorithms on one's own. The dartboard simulation used in this lesson provides a good example of a straightforward problem with a solution that someone can code on their own from scratch.

Guided Activity (45 min)

A Dartboard Simulation [10 min]

Get students' attention by asking them to play with the Dartboard Simulator (http://mw2.concord.org/public/student/game/dartboard.html) (requires the Java browser plug-in) as they think about the following scenario and answer the related questions:

- You have a circular dartboard consisting of three concentric circles. The largest outer circle has a radius of 3 units and is colored yellow. Inside that is the middle circle with a 2-unit radius, colored orange. In the center is a red circle, the bull's-eye, with a 1-unit radius. The final board looks like a red circle enclosed by an orange ring, enclosed by a larger yellow ring.
- Now suppose you have a robot that throws darts randomly. How many darts would you expect to hit the bull's-eye if the robot throws 90 darts at your dartboard, assuming every dart hits the board? How many of those darts (on average) would hit the orange part of the board? How many would hit the yellow part?
- To make the dartboard into a game, a certain number of points is awarded each time a dart hits a specific color. You want to assign score points to the dartboard fairly (for our random robot), such that the robot gets a score proportional to the chance it has of hitting a given color on the board. How many points should landing a dart on red be worth? On orange? On yellow?
- Question: What kind of a variable is needed to store the input for this program? Is the input to this program digital or analog? How do you know?
- Answers:
  - Recall that the formula for the area of a circle is A = pi*radius^2. The red circle has an area of pi units. The orange ring has an area of 3*pi units (4*pi minus the area of the red circle). The yellow ring has an area of 5*pi units (9*pi minus 3*pi for orange and pi for red). The total area is 9*pi. Thus, on average, 50 of the darts would land on yellow, 30 of the darts would land on orange, and 10 darts would land on red.
  - For score assignment, since a single dart would hit yellow 5/9 of the time, orange 3/9 of the time, and red 1/9 of the time, we know that red is 5 times harder to hit than yellow and 3 times harder to hit than orange. Thus, a convenient assignment would be 15 points for red, 5 points for orange, and 3 points for yellow. With these score assignments, if the robot throws a dart it would score 5 points on average.
  - This data is digital and can be represented with an integer variable. It only represents the final location of the dart. It doesn't supply any information about how it was thrown or other useful information that might be gathered while play a real dart game. This program ignores the details of the location and reduces the location to a single color area on the board. If exact position of the dart as it moved over time was

desired, then the analog data of location could be closely approximated digitally using a sampling technique, which means measuring values of the analog signal at regular intervals called samples. The samples are measured to figure out the exact bits required to store each sample.

Devising a Dartboard Simulation [10 min]

Suppose you want to write a program that simulates tossing virtual darts. Each dart will land at a point on a ***square*** virtual dartboard that is one unit long on each side. Each point on this dartboard has both an x and a y coordinate, both of which are between 0 and 1. The bull's-eye is a square in the center of the dartboard with sides of length 0.5 units.

Have the program ask the user how many darts they want thrown. The program should then simulate throwing these darts by generating a random landing location (a random x and a random y coordinate) on the dartboard for each dart. Recall how to use Python's random functions (by reviewing the previous lessons' dice simulation). As darts are thrown, the program counts how many darts land within the center square, the bull's-eye. The bounds of bull's-eye are [0.25, 0.75] on both the x and y axes. Finally, the program should print out the number of darts thrown and the number that landed within that rectangle.

Have your students answer the following questions:

1. Is the problem description clear enough that you could code this program on your own? If so, what information helps to make it clear? If not, what further explanations are needed?
2. What are some variables you think your program will need, and what will you use them for?
3. Are there any built-in Python functions that might be necessary or useful? If so, which ones?
4. What other types of Python constructs are required for this program? Specifically, will you need an input assignment? A loop? An if statement? A print statement? What else?
5. If you do need to use a loop, would it be better to use a while loop or a for loop in this program? Or will either work equally well? Why? If you think you can code the program without using a loop, explain why.
6. Think back to the dice simulation you made. What are two or three ways in which your dartboard program will be similar to the dice simulation? What are two or three ways they will be different?
7. What percentage of darts thrown randomly at the dartboard will hit the bull's-eye?

Programming the Simulation [25 min]

Take the rest of the class time to have your students begin programming their simulation. If they are not able to finish before the session ends, you may want to assign the program as homework, or devote the beginning of the second session to finishing the program. They will need their programs for the work in the next session.

You may want to remind students how to use Python's random function. The following code may be a useful example:

Example random coin flipping code (the python file is available in the Lesson Resources folder):

```
import random # Needed for random number generation

number_of_heads = 0
```

```
for i in range(0, 100):

    x = random.random() # Generates a random floating point (decimal) number
   between 0 and 1

    if x > 0.5:

        number_of_heads = number_of_heads + 1

print "The number of heads in 100 coin flips is ", number_of_heads
```

Optional Homework

Have your students finish their dartboard programs, as they are needed in the next session. Alternatively, if they have finished their programs, you could assign the "Collecting and Analyzing Data" think-pair-share of the next session as a homework, to be discussed in the next session.

Session 2

Getting Started (5 min)

Journal: Making your programs extensible

- Define "extensibility" for your students in the context of a program. Focus on the notion that we should plan for the future by keeping our code simple and organized logically. By doing so, we can more easily extend our code (add new features), reuse parts of our code in other programs, and more swiftly modify our program if the requirements change.
- Have your students answer the following questions in their journals: "What are some reasons to make our programs extensible? What are some things that can go wrong if our code is not extensible?"

Guided Activity (45 min)

Collecting and Analyzing Data [10 min]

Think-pair-share: Collecting and analyzing data

1. Ask your students to recall what percentage of darts randomly thrown will hit the bull's-eye.
2. With the completed square dartboard program, have your students collect data by running their program entering increasing values for the number of darts to be thrown (1, 10, 100, 500, 1000, 5000, 10000, 50000, and so on), recording the output for each input (number of darts that hit the bull's-eye). For each pair of input and output, have them calculate the percentage that hit the bull's-eye (divide output by input, e.g., 12 hits / 50 thrown = 0.24, or 24% of the darts were hits).
3. Have your students pair off and compare their data with that of their partner. Ask them to describe why the trends they observe in the data are there. Have them recall when they calculated the probability of the robot throwing darts at a circular dartboard.
4. Gather your students and discuss as a class what can be observed from the data. Ideally, as you increase the sample size (number of darts thrown), you should be getting a more accurate measurement of how many darts hit the bull's-eye. As with the dart-throwing robot, the area of the center relative to the whole dartboard determines the percentage that hit. In this case, the center square has an area of 0.25 units^2, and the rest of the dartboard

has an area of 0.75 units^2, so approximately 25% of the randomly thrown darts will hit the bull's-eye.

5. Expand the discussion to consider whether or not this is a realistic or appropriate model for a real life situation. What factors would make it more realistic? (changing levels depending on the skill of the thrower?, using data from a real life dart game to determine precentages?, entering values for angle and force to determine the path of a dart?, have students brainstorm). Point out that the suitability of the simulation depends on how it will be used.

Changing Requirements [30 min]

Discuss with your students how we often want to reuse our code for a new project, and how it is not uncommon when developing a program for the requirements to change. Both of these changes benefit from extensibility in code. Your students will get to test the extensibility of their dartboard program by reusing what they have to fit with a new objective: make a three-ringed circular dartboard. Point out that an iterative design process often starts with a simpler problem and then generalizes or extends that program to apply to other situations.

As before, this program should first ask a user how many darts they would like to throw. Then, it should use that input to simulate throwing darts at a circular dartboard. Finally, it should print the number of darts thrown, the number of darts that hit the bull's-eye, the number that hit the middle ring, the number that hit the outer ring, and the number that missed completely. This circular dartboard is similar to the one from the previous session's robot exercise: it has a central circular bull's-eye surrounded by a middle ring, which is in turn surrounded by an outer ring. The coordinates for this dartboard are: the center is at coordinate (0,0); the outermost ring is a circle with radius of 3; the middle has a radius of 2; and the bull's-eye has a radius of 1. Simulate throwing a dart by picking random x and y coordinates, each between -3 and 3. Since this range is a square, some darts may miss the dartboard completely. For this program, students may reuse as much of their square dartboard code as they need, but make sure to preserve their original program separately.

Homework

Have your students finish their circular dartboard programs and answer the following questions:

1. How extensible was your square dartboard simulation? How much code were you able to reuse for your circular dartboard simulation, and what did you need to write from scratch?
2. What are the major differences between the code in the programs and the way you approached coding them? Which did you find more difficult to program? Would it have been easier or faster to code the circular dartboard program from scratch?
3. Collect data by running the circular dartboard program with various increasing inputs (1, 10, 100, 500, 1000, 5000, and so on) and record the output. For each input and corresponding output, calculate the percentage of darts that missed, that hit the outer ring, that hit the inner ring, and that hit the bull's-eye. What trends do you see in the data? How is the behavior similar and different to the square dartboard program?

Options for Differentiated Instruction

Note: graphics.py may be used with this lesson to create a visualization - http://mcsp.wartburg.edu/zelle/python/ppics2/code/graphics.py (http://mcsp.wartburg.edu/zelle/python/ppics2/code/graphics.py)

Another Note: In order to run graphics.py, you may possibly need to save files in the same Python folder as the program being written due to student access and student space restrictions in your network, (i.e. if the link cannot point back to a network drive file due to restrictions).

Evidence of Learning

Formative Assessment

The students will produce two simulation programs on their own: the square dartboard simulation and the circular dartboard simulation.

Summative Assessment

The students will record their understanding of extensibility in their journal.

The students will gain experience collecting data in both sessions.

The students will think analytically about their programs by answering the questions in Session 1 and in the homework for Session 2.

(http://www.umbc.edu/) (http://www.umd.edu/)

(http://www.nsf.gov/)

Authored by: CS Matters in Maryland
Website: csmatters.org (http://csmatters.org)
Email: csmattersinmaryland@gmail.com (mailto:csmattersinmaryland@gmail.com)