Unit 1 Bootstrap Project Overview

http://www.bootstrapworld.org/materials/spring2016/courses/bs1/units/unit1/index.html

Unit 1 of the Bootstrap curriculum officially takes 85 minutes however we have created a revised version that will be 50 minutes.

Unit 1 mainly consists of three different stages. Together these stages introduce computational thinking on how video games are built along with how the different elements on the screen interact.

Stage 1 - *"Dissecting a Demo with X,Y Coordinates" 10min*

Each student will be asked in the preceding class to bring a picture of a scene in a video game they like or have played. To start we will all be using the NinjaCat video game (http://www.wescheme.org/run?publicId=sggzRzgU5T). The students will be asked to identify what characters exist in the screenshot of the NinjaCat game. All the characters will be listed in a table and we will add to the table what changes with every character when the game happens. Specifically we will focus on how each character changes considering X,Y coordinates. For example: Can NinjaCat move up and down in the game? Can she move left and right? So what's changing: her x-coordinate, her y-coordinate, or both? What about the clouds? Do they move up and down? Left and right? Both?

Stage 2 - *"Order of Operations for Numbers" 20min*

Math is a language, just like English, Spanish, or any other language. We use nouns, like "bread", "tomato", "mustard" and "cheese" to describe physical objects. Math has ***values***, like the numbers 1, 2, or 3, to describe quantities.

Humans also use verbs like "throw", "run", "build" and "jump" to describe operations on these nouns. Mathematics has ***functions*** like addition and subtraction, which are operations performed on numbers. Just as you can "slice a piece of bread", a person can also "add four and five".

Mathematicians didn't always agree on the order of operations, but now we have a common set of rules for how to evaluate expressions. The pyramid on the right summarizes the order. When evaluating an expression, we begin by applying the operations written at the top of the pyramid (multiplication and division). Only after we have completed all of those operations can we move down to the lower level. If both operations are present (as in 4+2−1), we read the expression from left to right, applying the operations in the order in which they appear.

Let's code!

We will be using WeScheme for our coding section. WeScheme is a free website that can be accessed on any computer with internet access. The WeScheme editor has a 'definition' and an 'interaction' section. You define your function in the 'definition' section and call it in the

'interactions' section with your desired values to perform the needed computation. http://www.wescheme.org/openEditor

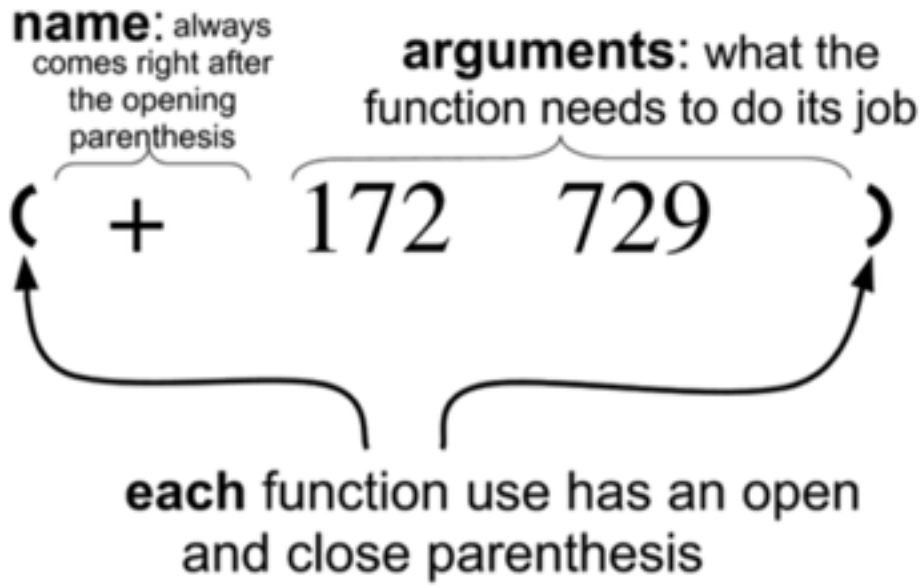
The Circles of Evaluation are also easy to convert into computer programs. To translate a Circle of Evaluation into a program, begin with an open parenthesis `(`, and then the function written at the top of the circle. Then translate the inputs from left to right in the same way, adding a closing parenthesis `)` when you're done. This process gives us the second rule for ***expressions***:

Code Rule 2: Each open parenthesis is followed by one function, then by one or more legal expressions, and finally by a closing parenthesis.

Here is the code for this Circle of Evaluation: `(- 4 5)`

See what happens when this code is entered into the Interactions area. Press the Return key to evaluate the program. You should see -1 as an answer.

All of the expressions that follow the function name are called *arguments* to the function. The following diagram summarizes the shape of an expression that uses a function.



Stage 3 - *“Order of Operations for Shapes” 20min*

The Circles of Evaluation are a powerful tool, and can be used for much more than just numbers. Consider the Circle of Evaluation shown here.

- What is the name of the function being used?
- How many arguments are being given to that function?
- What do you think this function will do?

The same rules you used to convert a Circle of Evaluation into code still apply. Here is the code for that Circle:

```
(star 50 "solid" "red")
```

Copy the above code into your definitions section. What happens when you click ‘Run’? What happens if you change the number to 100? What happens if you change the second argument in your function to “outline”?

See what happens if you change ‘star’ with other shapes like ‘circle’ or ‘square’?