Coding Concepts

GW Bootcamp 6/15/19

Computers Are Dumb!
(and also kind of like toddlers)

Computers read a set of instructions literally and output something, but they can't interpret those instructions- they can only follow them exactly as they're written.

- If you don't explicitly teach your computer a new word or command, it won't know what it means
- Computers forget things, unless you reinforce them (i.e. if you define a variable in one program, and you want to use it in another program, you have to remind the computer what the variable means before you can use it in a different program)
- If you give the directions out of order, a computer won't be able to figure out your intended order
- If you forget a comma or a period or a parenthesis or you improperly capitalize something, a computer won't be able to read the instruction properly
- If a computer can't understand something, or doesn't know how to do something, it will throw a big tantrum.

How Do Computers Read Instructions?

- Unidirectionally from top to bottom
- Unidirectionally from left to right
- Using strict grammatical rules
- As though they've never seen any of your custom commands/ variables before

Syntax

Syntax refers to those strict grammar rules that computers follow. The syntax will vary between languages, but some common elements are:

- "Balanced Parentheses" (a close parenthesis for every open parenthesis or bracket or curly bracket)
- Separators (e.g. commas between items, enters between commands, spaces in some places but not in others)
- Capitalization (for recognition of keywords and also consistency with variable names)

Types

Types are categories of coding objects that determine how the computer can use the objects within/ what the computer can do with them.

Different types have different rules associated with them (both in terms of syntax and in terms of use)

If you want to perform an operation on objects with different types, you often have to explicitly change the type of one of the objects (cast it) in order for the computer to

'string'	1	1.50	(1,2,3,4,5)
STRING	INTEGER	DOUBLE	ARRAY

STRING
- For text
- Can combine or shorten words
- Can access letters

INTEGER
- For whole numbers
- Can perform mathematical operations

DOUBLE
- For Decimals
- Can perform mathematical operations

ARRAY
- For storing information
- Can contain many different variables or numbers or strings

Variables

Computers will perform tasks, and then immediately forget them. If you want to be able to access information later on in your code, you have to store it in a variable.

Variable	=	Stuff Stored in Variable

Left side of the equal sign:
Where you're storing the Information (variable)

Right side of the equal sign:
The information that you're storing

Keywords

These don't really types, nor are they functions, but they are a set of words that the computer has been taught how to understand/ respond to.

For example, if a computer sees "IF" it knows that a conditional is coming. If a computer sees "For" it knows that a for loop is coming

*IMPORTANT: You should never use these keywords or built-in function names as variable names, because that will overwrite the rules associated with those words and they will no longer work in their intended manner

Functions

Input Stuff

Do Stuff

Output Stuff
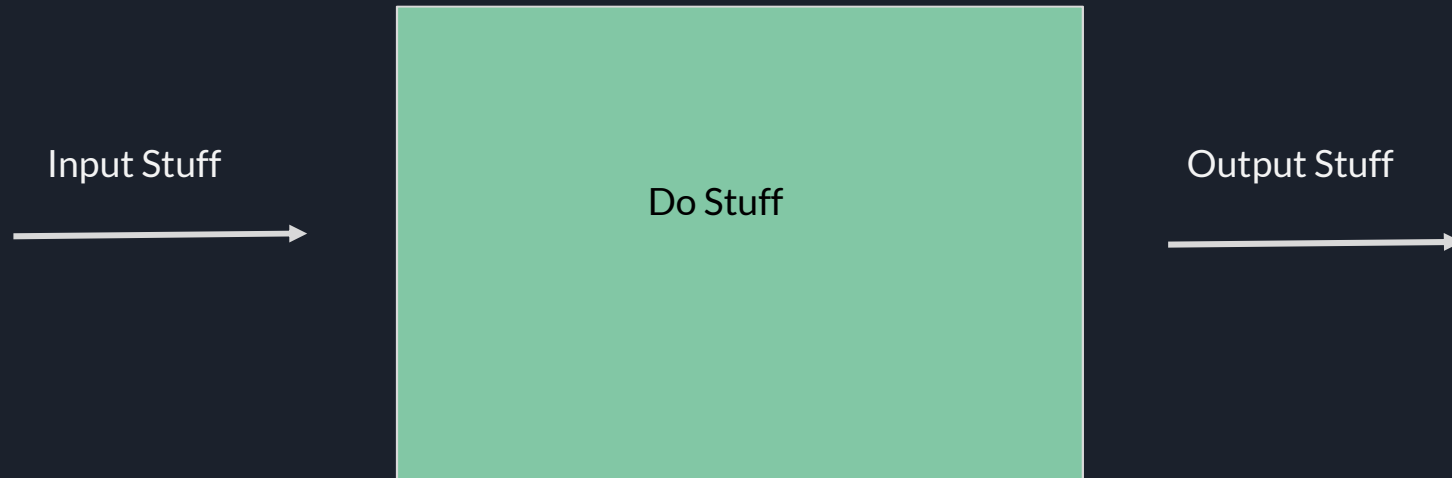
Functions are kind of like factories. Once you build the machines and conveyor belts and such inside, you can put many different things into your function, and it will perform the same set of procedures on those things and spit out the result.

For Loops

For loops are when you want to repeat a task a given number of times

For X being each item in a list:

    Do something to X

*Note: Often this list is a list of numbers, which equates to "Do Something X # of times"

*IMPORTANT: Anything you want to change goes inside your loop and anything you want to have stay the same goes outside your loop

While Loops

Pretty much the same thing as for loops, except they run until a certain condition is met, rather than for each item in a list

While X satisfies a condition:

      Do things to X

Sarah's Preferred Method for Writing For Loops:

1. Write out the task that you're trying to repeat for just one run of the task (e.g. if you're trying to modify a range of cells, write out the code for modifying just one cell)
2. Once you get it to work for one cell, write out the task for a second cell
3. Note which lines of code you had to edit to make it work for a second cell, and which lines of code you didn't have to touch
4. Put the lines you had to change inside of your for loop and the lines you didn't have to change outside of your for loop
5. Generalize your lines inside the for loop to work for the range of cells, instead of just the two that you did, using the variable in the "FOR" line

Conditionals

I feel like we're pretty good on conditionals, but as a reminder:

IF   If A condition is met:

                Do X

ELSEIF   If A condition is not met, but B condition is met:

                Do Y

ELSE   If none of the conditions are met:

                Do Z

Note: An "IF" statemet automatically restarts the process, so if you have two "IF"s, instead of an "IF" and an "ELSEIF," and both conditions are met, then both tasks will be performed. If you have an "IF" and an "ELSEIF," and both conditions are met, oly the "IF" clause will be performed

Error Messages

We are conditioned to see red ink as a sign of failure, but in coding it's actually a really good thing, because it's feedback from our computer about how it's reading our code. Remember, computers are dumb, so we can never assume that they're interpreting our commands the way we want them to be interpreted.

I think we'll talk more about error messages today or in a future class, but they often provide a lot of information, including where the "error" is in your code and what the computer doesn't understand about it (consider errors misunderstandings rather than you doing something wrong)

Debugging= finding and clarifying or correcting those misunderstandings

GOOGLE IS YOUR FRIEND!!! I cannot stress that enough. Even the most talented and experienced programmers Google error messages all the time.

Debugging Techniques

- Read/ google (if necessary) your error message
- Count your open parentheses and your closed parentheses and make sure the numbers match up
- Compare your code to examples online or in the class activities folder, letter by letter, to see what looks different in your code vs. the example code
- Run your code one line at a time, starting at the top, to see where the error or unexpected result is first occurring. Attempt to fix that, then keep going down your program if there's still an issue.

A Note About My Teaching Style (and Dart's Too, but I Can't Speak for Him)

- Since coding is ALL ABOUT getting error messages or unexpected results and learning how to handle them, I will try very hard not to just point out what's wrong in your code and tell you how to fix it, but rather I will guide you through how to read error messages and respond to them, continuously, until you no longer get error messages.
- This may seem very frustrating, but please try to be patient, because being able to troubleshoot on your own is the most valuable thing you can take away from this class.
- My answer to "Will this work?" will almost always be "I don't know, test it," because when I'm writing code, I can't assume that it will work until I test it and see.
- Sometimes when I say "I don't know, test it," it's a teaching moment, but sometimes I genuinely don't know. There are many different ways to achieve the same result when you're coding, and so you might not be doing it the way I would've thought to do it, but that doesn't necessarily mean it won't work.