Vim class notes

Chris Wallace

Week 3

Recap: What did we talk about last week?

- file commands (touch, mv, cp, rm, etc…)
- folder commands (mkdir, rmdir, cd, etc…)
- input and output redirection (and cowsay, of course)
- the echo command
- variables

Variables

Let's talk again about variables, since they are actually used in almost every language!

How do we *statically assign* content to a variable?

```
name="chris"
```

How do we *dynamically assign* content to a variable?

```
read name
```

How do we print the contents of a variable?

```
echo "$name"
echo "$other_variable"
```

Variables are some of the most important things of programming. If you have taken pre algebra, you should be familiar with something like this:

$$y = x + 2$$

To represent this in bash, we first introduce that there is a way to do math: `$((math))`. See the example below, notice we use the "$" only when *accessing* a variable's contents.

```
x=2
echo "$x" # shows 2
y=$(($x+2)) # y is now 4
```

In programming, we use variables to keep track of *state*. We want our programs to be able to make decisions based on what they receive and know.

Conditionals

Almost every programming language has a thing called *Conditionals*. If you've done scratch or python, you've seen these already: "if this then that" is the general idea.

Let's look at some examples. In the following, we can see a simple **if** statement. Notice that the square brackets have some pretty specific spacing, and we use the $ to access the choice. Also important are the **then** statement, and the **fi** to end. We additionally indent our code for readability.

```
choice=0
if [ $choice -eq 0 ]
then
    echo "choice was rock!"
fi
```

In the following example, we'll see "if else". The **else** block will happen if the **if** condition was not true. What do you think this program will print?

```
choice=999
if [ $choice -eq 0 ]
then
    echo "choice was rock!"
else
    echo "choice was not rock! :("
fi
```

Finally, we have one more tool at our disposal: **elif**. We use this to handle *multiple* conditions. What do you think will be printed by this?

```
choice=1
if [ $choice -eq 0 ]
then
    echo "choice was rock!"
elif [ $choice -eq 1 ]
then
    echo "choice was paper!"
else
    echo "choice was not rock, or paper.. hmmmm"
fi
```

Rock Paper Scissors

In the case of the rock paper scissors project, we can make decisions based on what the user inputs. Before we do that, however, we need to discuss algorithms.

Algorithms

An *algorithm* is the set of rules, or steps, that the computer will take to solve a problem. As a software engineer, you'll have to not just type code into the computer, but actually come up with solutions that work.

For Rock Paper Scissors, can you think of the steps involved? Here's the simplest steps we came up with:

1. Both players come up with a choice.
2. A winner is determined.

Let's break down each individual part of that. in #1, we already know how to get human input using **read**, so the hard part is figuring out the computer's choice.

The Computer's Choice

Remember remainder division, your favorite part of elementary school math? There's one thing that's important from what you learned back then. Check out the following example, notice the pattern of the remainder:

$$\frac{2}{2} = 1, \text{remainder } 0$$
$$\frac{3}{2} = 1, \text{remainder } 1$$
$$\frac{4}{2} = 2, \text{remainder } 0$$
$$\frac{5}{2} = 2, \text{remainder } 1$$

Whenever we divide by 2, we always get a remainder of either 0 or 1.

What happens when we divide by 3? The possible remainders then become 0, 1, and 2.

$$\frac{0}{3} = 0, \text{remainder } 0$$
$$\frac{1}{3} = 0, \text{remainder } 1$$
$$\frac{2}{3} = 0, \text{remainder } 2$$
$$\frac{3}{3} = 1, \text{remainder } 0$$
$$\frac{4}{3} = 1, \text{remainder } 1$$
$$\frac{5}{3} = 1, \text{remainder } 2$$

In programming, most languages have a shortcut to get "the remainder from a division of this number". It's called *modulo*, or **mod** for short. You can use it in bash with: `%`, for example:

```
x=3
echo "$(($x % 3))" # prints 0
x=4
echo "$(($x % 3))" # prints 1
x=5
echo "$(($x % 3))" # prints 2
```

Now that we have a way to take any number to 0, 1, or 2, we need a way to get random numbers. Luckily, there is a *system variable* that gives you a random number for free. It's called `RANDOM`, simply enough. To use:

```
# get a random number between 0 and 2
computerchoice=$(($RANDOM % 3))
```

Determining a winner

To determine a winner, we'll use the **if** statements we learned earlier, by *nesting* the if statements (putting them inside each other):

```
if [ $playerchoice -eq 0 ]
then
    echo "player chose rock"
    if [ $computerchoice -eq 0 ]
    then
        echo "there was a tie"
        # you do the rest
    fi
elif [ $playerchoice -eq 1 ]
then
    echo "player chose paper"
    # you do the rest...
elif [ $playerchoice -eq 2 ]
then
    echo "player chose scissors"
    # you do the rest...
else
    echo "player didn't make the right choice"
fi
```