March 23rd Workshop Summary

Syntax Introduction

Introduction:

- Code is essentially series of statements that tells computers to do stuff
- However computers are unlike humans - they only recognize certain things. Therefore we need standard and exact syntax to communicate with computers. Otherwise they won't understand and will throw errors.
- Computers really only understand binary (0 and 1), but compilers understand many languages such as C, C++, Java. In this workshop, we'll be using Java. You can think of compilers as translators that will translate these different languages to the binary that computers understand.

Main Function:

How does the computer know where to start executing the list of things you want it to do? They will always be looking for "main function", which is a function that **must** look **exactly** like this:

```
5   public static void main(String[] args) {
6   // write your code here
7       System.out.println("hello world");
8   }
```

General Syntax:

Statements:

- Statements are like commands (i.e. "do this").
- Every statement needs to be closed by a semi colon, so your computer knows that you are done your command.

Things you can do:

1. **Printing** - prints a sentence to the console
   E.g.
   `System.out.println("hello world");` //prints "hello world" followed by a new line
   `System.out.print("hello world");` //prints "hello world" without the new line

2. **Variable Definition and Assignment** - create a variable of a specific type
   General variable declaration looks like: `<type> <variableName> = <value>;`
   E.g.

```
int myInteger = 3;          //creates an integer variable assigned with value 3
char myCharacter = 'a';     //creates a character variable of value 'a'
String myWord = "apple";    //creates a word/sentence of value "apple"
double myNumber = 3.14;     //creates a number that allows decimals of value 3.14
boolean myBoolean = true;   //creates a boolean of type true (or false)
```

note: computers need to remember the variables we have created. However every variable type takes up different amount of space in computer's memory (i.e. a sentence definitely takes up more memory than a character). In order for computers to know how much memory it needs to allocate, we need to tell it the type of the variable. Therefore to declare variables we want, we also need to write out the type before the name that we give the variable to.

3. **Operations** - mathematical operations, comparison operations
   E.g.

```
myInteger = myInteger + myInteger;
myNumber = 3.14 / 2;
int six = 2 * 3;
int meaningOfLife = 43 - 1;
boolean equates = (3 == 3) && (5 == 5);
boolean isNotLessThan = !(3 < 2);
boolean thisIsFalse = 4 >= 5;
boolean thisIsTrue = (2 > 3) || isNotLessThan;
```

   Other operations include `> (strictly greater than), <= (smaller or equal to), != (not equal to)`

Blocks:

- Blocks are like containers for statements.
- These containers, just like your drawers, are used to organize the statements
- By organizing these statements, you can wire your code logic

Things you can do with blocks:

1. **IF / Else statements** - will only execute statements if certain condition is satisfied
   E.g.

```java
int n = 10; //creates n and assigns it 10
if (n > 5) {
  //will execute if n is greater than 5
  System.out.println("n is greater than 5");
} else if (n > 3) {
  //will only execute if n is greater than 3, but first condition does not satisfy
  System.out.println("n is greater than 3, but smaller or equal to 5");
} else {
  //will execute if none of the previous conditions satisfy
  System.out.println("n is smaller or equal to 3");
}
//RESULT: prints out "n is greater than 5" ONLY

```

```java
int n = 10;
// will execute if n > 5
if (n > 5) {
  System.out.println("n is greater than 5");
}

// will execute (as well) if n is greater than 5
if (n > 3){
  System.out.println("n is greater than 3");
}
//RESULT: prints out BOTH "n is greater than 5" and "n is greater than 3"
```

The conditions inside the “if brackets” (i.e. n > 5) needs to have a ‘boolean’ type, since we only want the condition to be either true or false.

2. **Loops** - will execute statements as many times as you want

   For Loops has the following structure:

   ```
   for(<variable declaration> ; <condition>; <operation>){
         <expressions>
   }
   ```

   E.g.

   ```
   for(int i = 0; i < 10; i++){
         System.out.println(“*”);
   }
   ```

- In the example above, we have declared (created) variable i and assigned it to 0.
- The loop will only run under the condition, which is that i must be less than 10;
- Everytime after it executes the stuff inside the loop, i will increment by 1;
- The stuff it executes is printing a “*”;
- Therefore it would print it 10x, since it would print it when i = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

  While loop has the following structure:

```
while(<condition>){
      <expressions>
}
```
E.g.
```
int i = 0;
while(i < 10){
      System.out.println("*");
      i++;
}
```

- The above code does the exact same thing as the for loop example.
- We first declare the variable i outside of the while loop and assign it to equal 0.
- If the condition satisfies, in this case if i is strictly smaller than 10, we print out a "*" and increment i by 1;
- We keep executing in this fashion until i reaches 10.

3. **Functions** - will execute statements and blocks once function is called
   Function has the following structure:

```
public static <return-type> <function-name> (<type> <parameter name> ...) {
   <statements here>
}
```

E.g.

```
public static int addIntegers (int integer1, int integer2){
   return integer1 + integer2;
}
```

- 'public' and 'static' keywords are beyond the scope of our crash course, so let's just leave everything as public and static for now.
- In the example above, we see that we want to return an integer in this function, therefore our return type is 'int'.
- The name we want to give this function is "addIntegers", so that we know which function to call when we want to do a specific thing.
- We pass two parameters into this function, the first integer we want to add (integer1) and the second integer we want to add (integer2), and both needs a variable type declared beside it.
- Finally, we need to return the correct result we want the function to return, and the type of this result MUST match with the return type.

Variable Scope:

When we create variables, some variables are not accessible everywhere.

In most cases:

1. Variables declared within a block cannot be accessed from outside of the block.
2. But variables declared outside of the block can be accessed within the block, only if the blocks nests within each other.
3. Variables declared in a completely separate (exclusive) block cannot be accessed;

E.g.

```
public static void scopeTesting() {
	int myNumber = 10;
	if (myNumber == 10) {		//myNumber could be accessed due to rule #2
		String greeting = “hello!”;
	}
	System.out.println(greeting); //greeting CANNOT be accessed due to rule 1
}

public static void scopeTesting2(){
	int myNumber = 10;
	if(myNumber == 10){		//myNumber could be accessed due to rule #2
		double pi = 3.14;
	}else{
		double tau = pi + pi;	//tau CANNOT be accessed due to rule #3
	}
	for(int i = 0; i < 10; i++){
		i++;
		System.out.println(i);
	}
	System.out.println(i);	//i CANNOT be accessed due to rule #1
}
```

Additional Resources:

Don’t worry! Coding can get pretty tricky when you first start. However after just a little practice, you’ll get the hang of it :)
Here’re some additional resources for our crashcourse:

Github link: https://github.com/CSFoundations
We’ll be posting workshop summaries, solutions to our workshop problems under this link.
Slack link:
https://join.slack.com/t/csfoundations/shared_invite/enQtMzM1Nzg1NjE5MjgzLTM0NTMwOGE3YmUwOWZjMDFiOGIzYTY5NzViNTgxNjQ5OTg2MmU5Njk4OTkxOGQ1NTcxMGQyNWQzYzBjMTg4MWE
If you have any questions you can slack any one of us and we’ll answer ASAP :)