Variables and Constants

What are variables?

- The nouns of programming
- Named containers that hold data
- Data can be numbers, text, or more complex values
- Example: a user’s score in a game

Anatomy of a variable

- var keyword
- Variable name
- Variable data type
- Set value using =

Example of variable

```
var score:Int = 100
```

Example of constant

```
// if the var isn’t going to change, use let instead


let score:Int = 100


// after setting a value for a constant, you cannot change it
```

Rules for naming variables

- Start with a lowercase letter
- Use only letters, numbers, and underscores
- Use camelCase
- Don’t use reserved words (watch for color changes)

Optionals!

- Optionals *must* be used in Swift when a variable's value can possibly be null
- When an optional is read, the initial reading just checks if the var is nil or non-nil
- Must unwrap variables before the actual value can be accessed

Wrapping and Unwrapping Optionals

- When declaring an optional, use a question mark or exclamation point
- Question marks denote vars that you will unwrap on your own, again using a question mark
- Exclamation point optionals are called Implicitly Unwrapped Optionals

Example of optional

```
// creation

var myName:String?

// assignment

myName = “Todd”

// access

print("\(myName?.uppercaseString)")

// if myName is null on the above line, code execution after ?
will be ignored
```

Example of implicitly unwrapped optional

```
// creation

var myName:String!

// assignment

myName = "Todd"

// access

print("\(myName.uppercaseString)")

// if my name is null on above line, app will crash
```

Functions/Methods

What are functions?

- The verbs or actions of programming
- Named sections of code that are logically grouped
- Reduce verbosity and redundancy in code
- Like any real world action, definition is not execution

Anatomy of a function definition

- func keyword
- Function name
- Body of function, usually multiple lines of code
- Function body is wrapped in curly braces { }

Anatomy of a function execution

- Name of object executing the function (normally self)
- Function name
- Parentheses, optionally containing parameters

Example of a function

```
self.tieShoes()

func tieShoes()
{
    // code that runs when tieShoes is called
}
```

Rules for naming functions

- Start with a lowercase letter
- Use only letters, numbers, and underscores
- Use camelCase
- Don’t use reserved words. (Watch for color changes)

Parameter Functions

What are parameter functions?

- Way for functions to have more versatility
- Can provide different input for same function
- Perform the same process on different input
- Input is called a parameter, much like a variable

Anatomy of a parameter function

- Parameters are added in parentheses
- Parameter name, colon, data type
- Multiple parameters are separated by commas
- Executed function only names parameters after the first

Example of a parameter function

```
self.exercise(30)

self.exercise(60)

func exercise(numberOfMinutes:Int)
{
    // numberOfMinutes can be used like a variable
    // but only within this function
}
```

Rules for naming functions

- Start with a lowercase letter
- Use only letters, numbers, and underscores
- Use camelCase
- Don’t use reserved words. (Watch for color changes)

Returning Data From Functions

Returning data

- Can use function to run a task that gives a certain output
- Declare the data type using the **->** operator
- Void return data type is used if none is declared

Example of returning data

```
func exerciseFor(numberOfMinutes:Int,daysPerWeek:Int) -> Int
{
    return numberOfMinutes * daysPerWeek
}

let total:Int = self.exerciseFor(30,daysPerWeek:5)

print("you exercised for \(total) minutes this week")

// will print you exercised for 150 minutes this week
```

Conditional Statements

What are conditional statements?

- If/then statements
- A condition is evaluated as true or false
- A line or section of code is executed based on the result

Anatomy of a conditional statement

- if (condition)
- Section of code to execute if true is in curly braces
- Can optionally be followed by else statement
- else statement runs only if condition is evaluated as false

Example of a conditional statement

```
if(1 > 3)
{
    // code that runs if condition is evaluated as true
}
else {
    // code that runs if condition is evaluated as false
}
```

Conditional operators

Operator	Meaning	Example
==	is equal to	if (1 == 1)
>	greater than	if (2 > 1)
<	less than	if (1 < 2)
&&	and	if (1 == 1 && 2 > 1)
||	or	if (1 == 0 || 1 == 1)
!=	NOT equal to	if (1 != 0)

Arrays

What are arrays?

- lists of items
- organized by numbers starting at 0 (not 1)
- effective way to keep content in a particular order

Example workflow of an Array

- declare the array variable, including data type
- create the array, shorthand or longhand
- optionally add/change elements in the array
- access array elements shorthand or longhand

Example of an Array

```
// shorthand creation

var names:[String] = ["Todd","Ted","Tad"]


// access array elements (result would be Ted)

print("\(names[1])")
```

Loops

What are loops?

- run a section of code multiple times
- useful to check multiple items in an array for a given condition
- run instantly, between frame updates, not over time

Example workflow of a loop

- create the loop using the for or repeat keyword
- specify the condition that needs to be met for the loop to stop
- add the code that is executed with each iteration of the loop

Example of a for loop

```
var names:[String] = ["Todd","Ted","Tad"]


for (var i:Int = 0; i < names.count; i++) {
    print("\(names[i])")
}
// will print:
// Todd
// Ted
// Tad
```

Example of a repeat while loop

```
var names:[String] = ["Todd","Ted","Tad"]


var i:Int = 0


repeat {
    print("\(names[i])")
    i++
} while (i < names.count)


// will print:
// Todd
// Ted
// Tad
```

Classes

What are classes?

- Similar to a blueprint, explains the functionality and attributes of a user-created code object

- Think of a dog as a class. Dogs have certain attributes (hair, legs, tail, etc.) and perform certain actions (barking, tail wagging, chewing on your stuff)

Example of creating a class

```
class Dog {
    var weight:Int!

    func bark() {
        if self.weight < 10 {
            print("yap yap")
        }
        else {
            print("bark bark")
        }
    }
}
```

Example of using a class

```
let myDog:Dog = Dog()

myDog.weight = 11

myDog.bark()
```

Command Line Utilities

What are command line utilities?

- Can be made from Xcode templates (OS X > Command Line Utility)

- Useful for writing scripts, because you get code hinting/ coloring, but can still run from a command prompt

Classes vs. main.swift

- main.swift acts as the entry point for your app
- Only place you can run top level code
- For more info, see https://developer.apple.com/swift/blog/?id=7 (Swift Blog - Files and Initialization)

Running Swift from the Command Line

Understanding Command Line Swift

- Three ways to use Swift in the command line
- Directly (type Swift code into Terminal)
- As a script
- As program
- As a binary

Running Swift Directly

- Simply type swift in Terminal
- Type Swift code into Terminal, one line at a time
- See commands by calling :help
- Can exit out of this mode using :exit

Running Swift as a script

- Navigate to appropriate directory
- swift filename.swift
- Can pass in arguments from the command line here (ex. swift filename.swift "argument example")
- Cannot use multiple files natively, so you have to create your own workarounds

Run as a program

- Add shebang to swift file (#!/usr/bin/env swift)
- Make sure to change permissions on the file (chmod +x filename.swift)
- Execute the file (./filename.swift)

Compiling and running

- Compile swiftc (swiftc filename.swift -o binaryname)
- Run as usual (./binaryname)

Working with NSTask

What is NSTask

- Run command line code from a GUI
- Effective for simplifying a process

Using NSTask

- Create the NSTask object
- Set the command you want to use via the launchPath property
- Set arguments property as a [String]
- Launch the task
- Optionally call waitUntilExit() to make sure task is done before next one starts

Example of using NSTask

```
let task:NSTask = NSTask()

task.launchPath = "/usr/bin/say"

task.arguments = ["-v","Whisper",textField.stringValue]

task.launch()

task.waitUntilExit()
```