GENERAL ASSEMBLY

MOBILE DEVELOPMENT

INTRO TO SWIFT

Rudd Taylor
Founder, SALT

LEARNING OBJECTIVES

- Define Swift and its value to the iOS ecosystem
- Define and demonstrate playgrounds
- Define Swift's fundamental data types
- Use variables and constants, and understand the difference between the two
- Apply optionals and understand when to use them
- Utilize control flow to create a simple program flow in playgrounds

REVIEW ASSESSMENT AND

LAST CLASS

REVIEW QUESTIONS

- What is a View Controller?
- Why are segues important and how do you use them?
- Define Navigation Controllers and give a sample use case.

INTRO TO SWIFT

ABOUT SWIFT

- iOS (7+) and OS X (Mavericks+) development
- Object-Oriented
- Compiled
- “Safe”
- Playgrounds
- Works with Objective-C

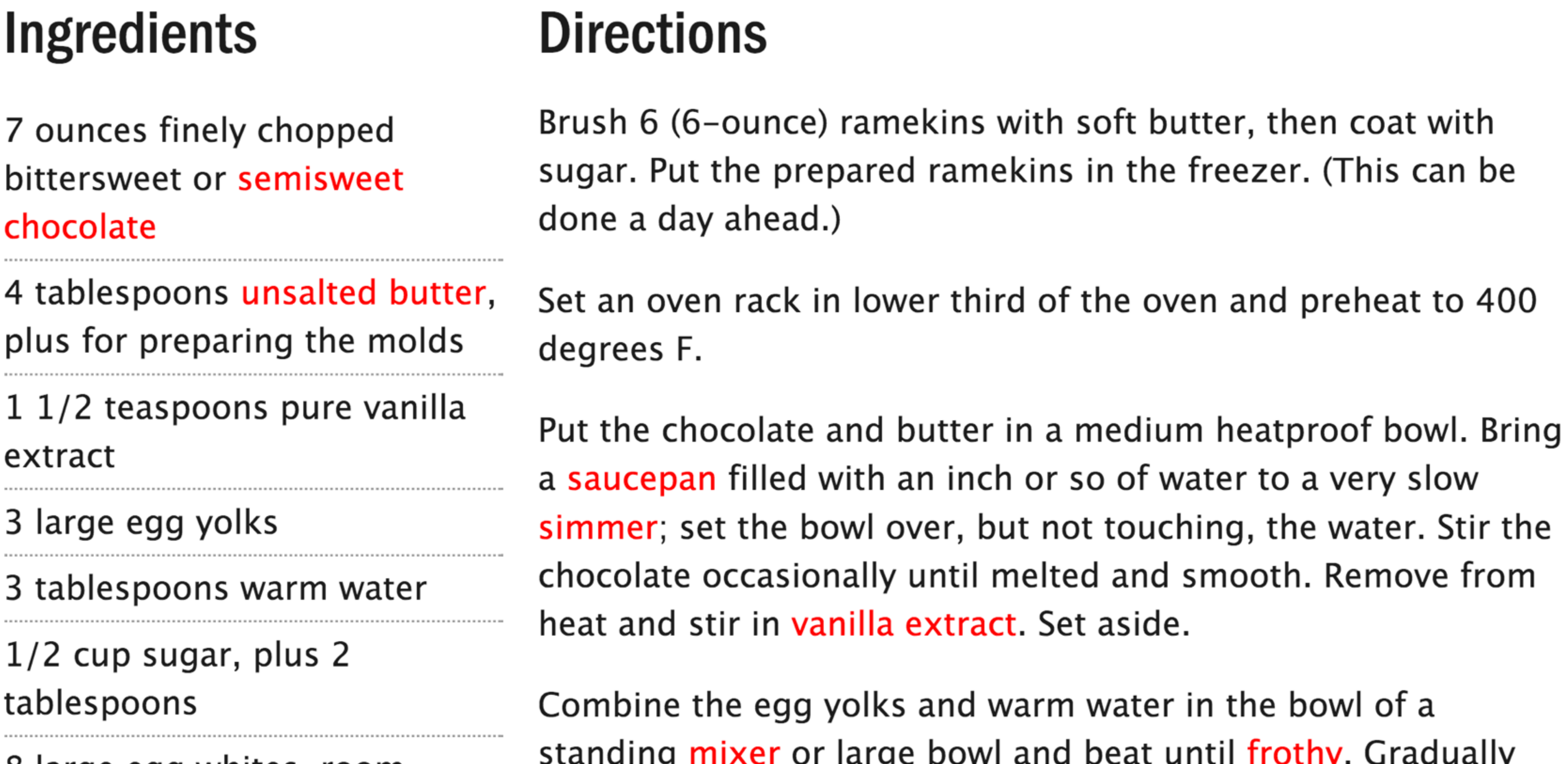
INTRO TO SWIFT

SWIFT VS

HOW CODE IS EXECUTED

- Our code is like a recipe for a meal.
- The computer will start with the first instruction, complete it…
  - Then move on to the second instruction, complete that…
  - Repeat until it is done with instructions.
- Unlike a recipe, we have to be much more specific with computer code.
  - Computers are fast and dumb.
  - i.e. They will do exactly what you say, mistakes and all. (Although sometimes apps seem like they often have minds of their own.)

LIKE A RECIPE – CHOCOLATE SOUFFLE



Ingredients

- 7 ounces finely chopped bittersweet or semisweet chocolate
- 4 tablespoons unsalted butter, plus for preparing the molds
- 1 1/2 teaspoons pure vanilla extract
- 3 large egg yolks
- 3 tablespoons warm water
- 1/2 cup sugar, plus 2 tablespoons
- 8 large egg whites, room

Directions

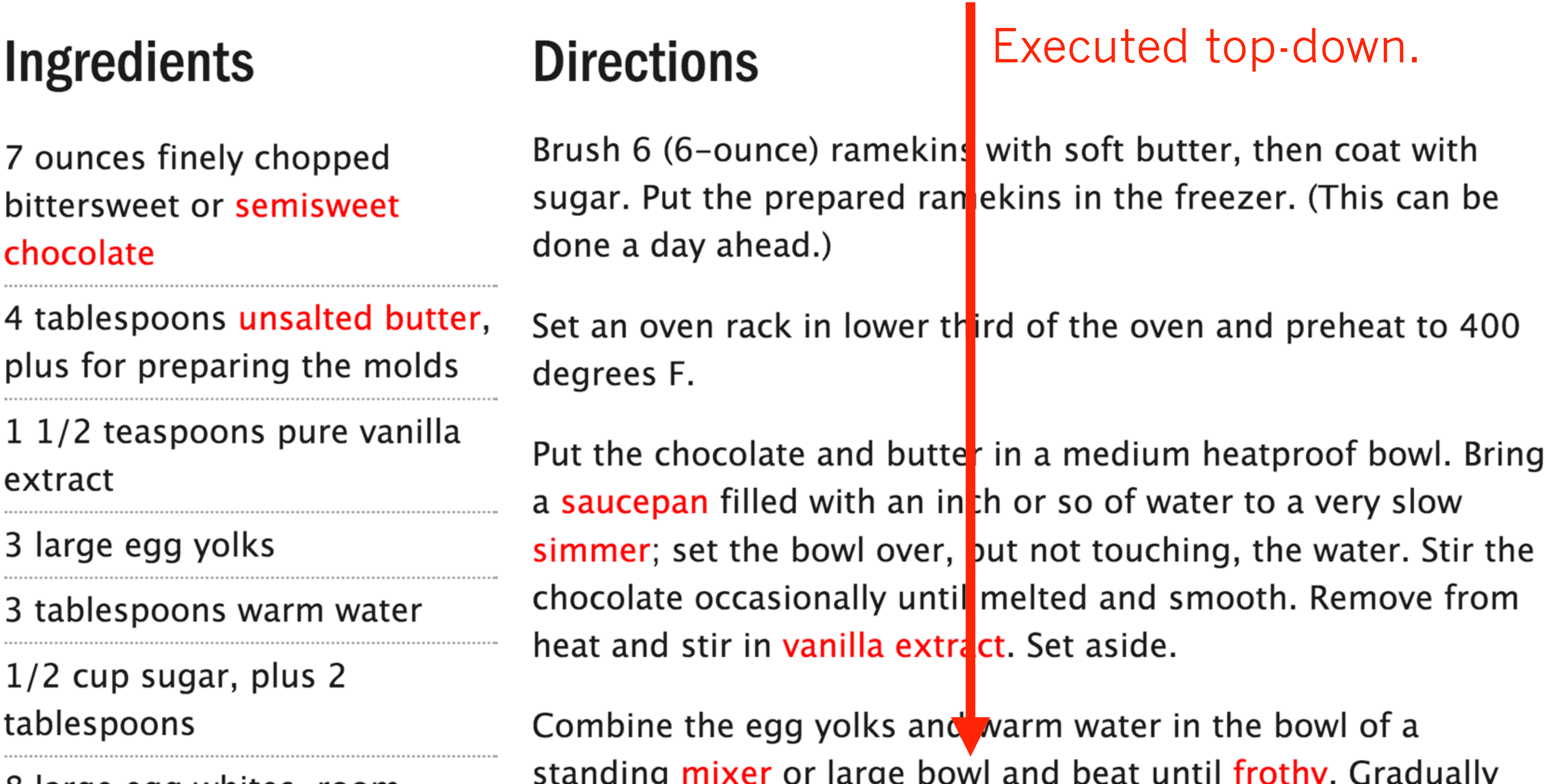
Brush 6 (6-ounce) ramekins with soft butter, then coat with sugar. Put the prepared ramekins in the freezer. (This can be done a day ahead.)

Set an oven rack in lower third of the oven and preheat to 400 degrees F.

Put the chocolate and butter in a medium heatproof bowl. Bring a saucepan filled with an inch or so of water to a very slow simmer; set the bowl over, but not touching, the water. Stir the chocolate occasionally until melted and smooth. Remove from heat and stir in vanilla extract. Set aside.

Combine the egg yolks and warm water in the bowl of a standing mixer or large bowl and beat until frothy. Gradually

LIKE A RECIPE – INSTRUCTIONS



Executed top-down.

Ingredients

- 7 ounces finely chopped bittersweet or semisweet chocolate
- 4 tablespoons unsalted butter, plus for preparing the molds
- 1 1/2 teaspoons pure vanilla extract
- 3 large egg yolks
- 3 tablespoons warm water
- 1/2 cup sugar, plus 2 tablespoons
- 8 large egg whites, room

Directions

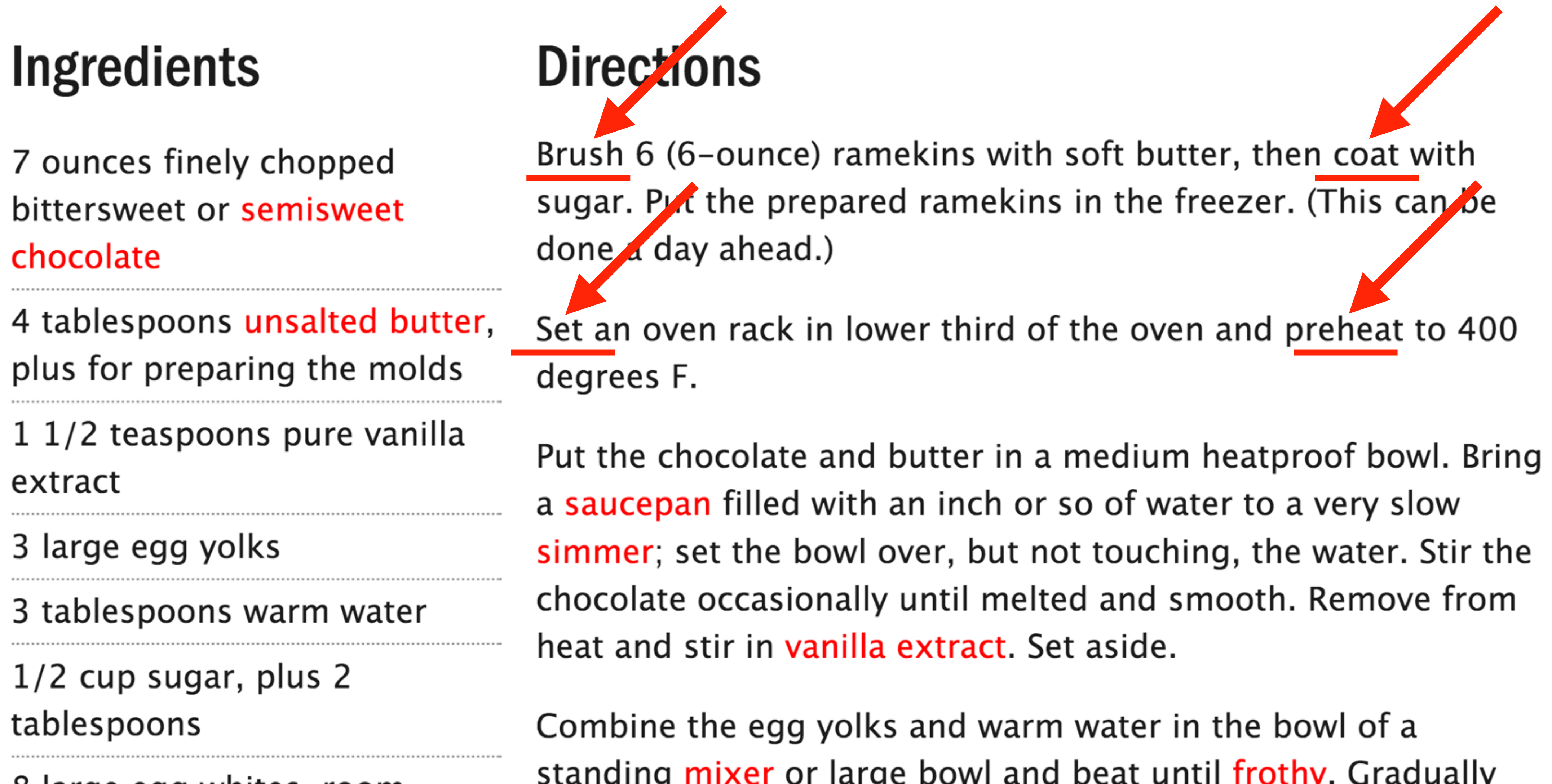
Brush 6 (6-ounce) ramekins with soft butter, then coat with sugar. Put the prepared ramekins in the freezer. (This can be done a day ahead.)

Set an oven rack in lower third of the oven and preheat to 400 degrees F.

Put the chocolate and butter in a medium heatproof bowl. Bring a saucepan filled with an inch or so of water to a very slow simmer; set the bowl over, but not touching, the water. Stir the chocolate occasionally until melted and smooth. Remove from heat and stir in vanilla extract. Set aside.

Combine the egg yolks and warm water in the bowl of a standing mixer or large bowl and beat until frothy. Gradually

LIKE A RECIPE – INSTRUCTIONS



Ingredients

7 ounces finely chopped bittersweet or semisweet chocolate

4 tablespoons unsalted butter, plus for preparing the molds

1 1/2 teaspoons pure vanilla extract

3 large egg yolks

3 tablespoons warm water

1/2 cup sugar, plus 2 tablespoons

8 large egg whites, room

Directions

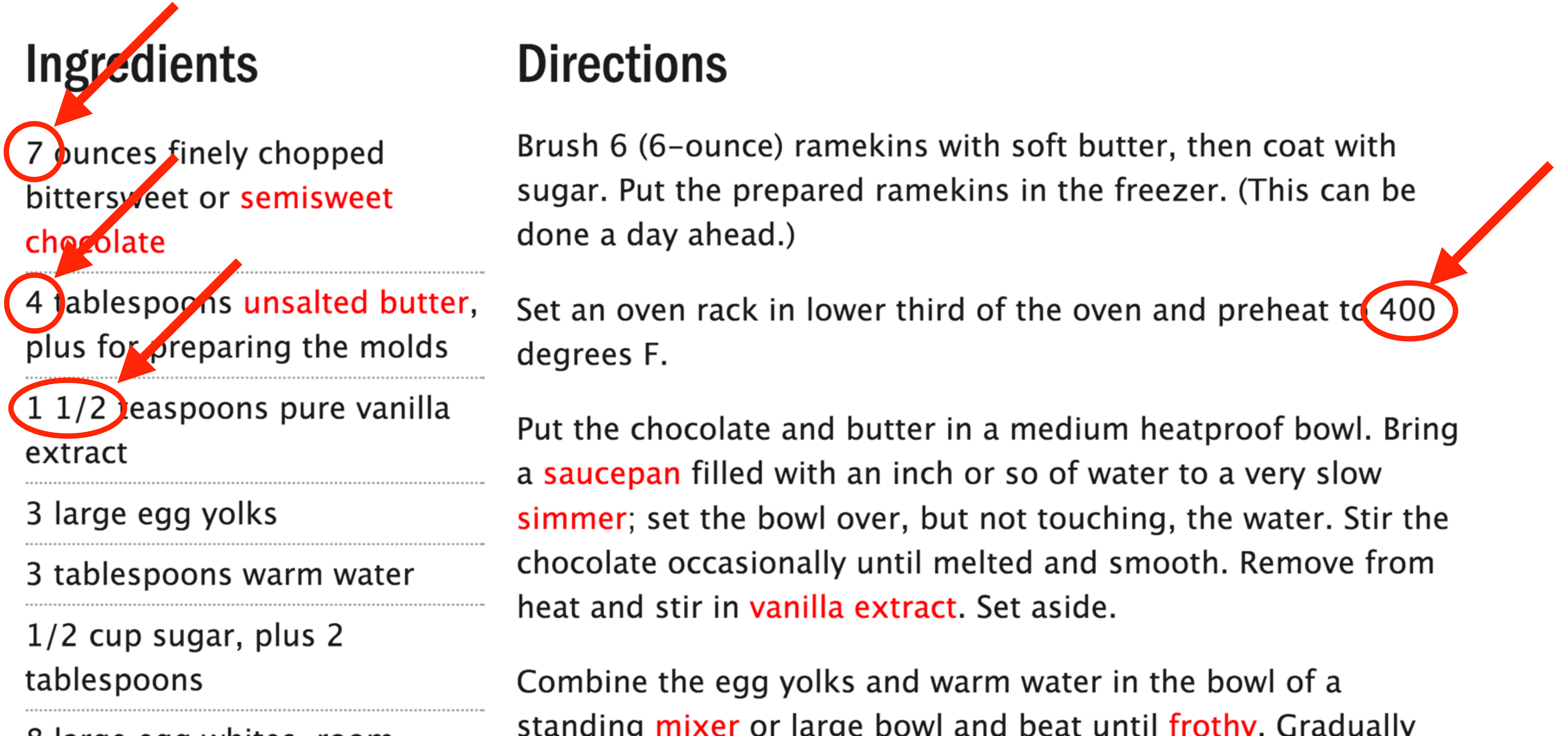
Brush 6 (6-ounce) ramekins with soft butter, then coat with sugar. Put the prepared ramekins in the freezer. (This can be done a day ahead.)

Set an oven rack in lower third of the oven and preheat to 400 degrees F.

Put the chocolate and butter in a medium heatproof bowl. Bring a saucepan filled with an inch or so of water to a very slow simmer; set the bowl over, but not touching, the water. Stir the chocolate occasionally until melted and smooth. Remove from heat and stir in vanilla extract. Set aside.

Combine the egg yolks and warm water in the bowl of a standing mixer or large bowl and beat until frothy. Gradually

LIKE A RECIPE – VALUES



Ingredients

7 ounces finely chopped bittersweet or semisweet chocolate

4 tablespoons unsalted butter, plus for preparing the molds

1 1/2 teaspoons pure vanilla extract

3 large egg yolks

3 tablespoons warm water

1/2 cup sugar, plus 2 tablespoons

8 large egg whites, room

Directions

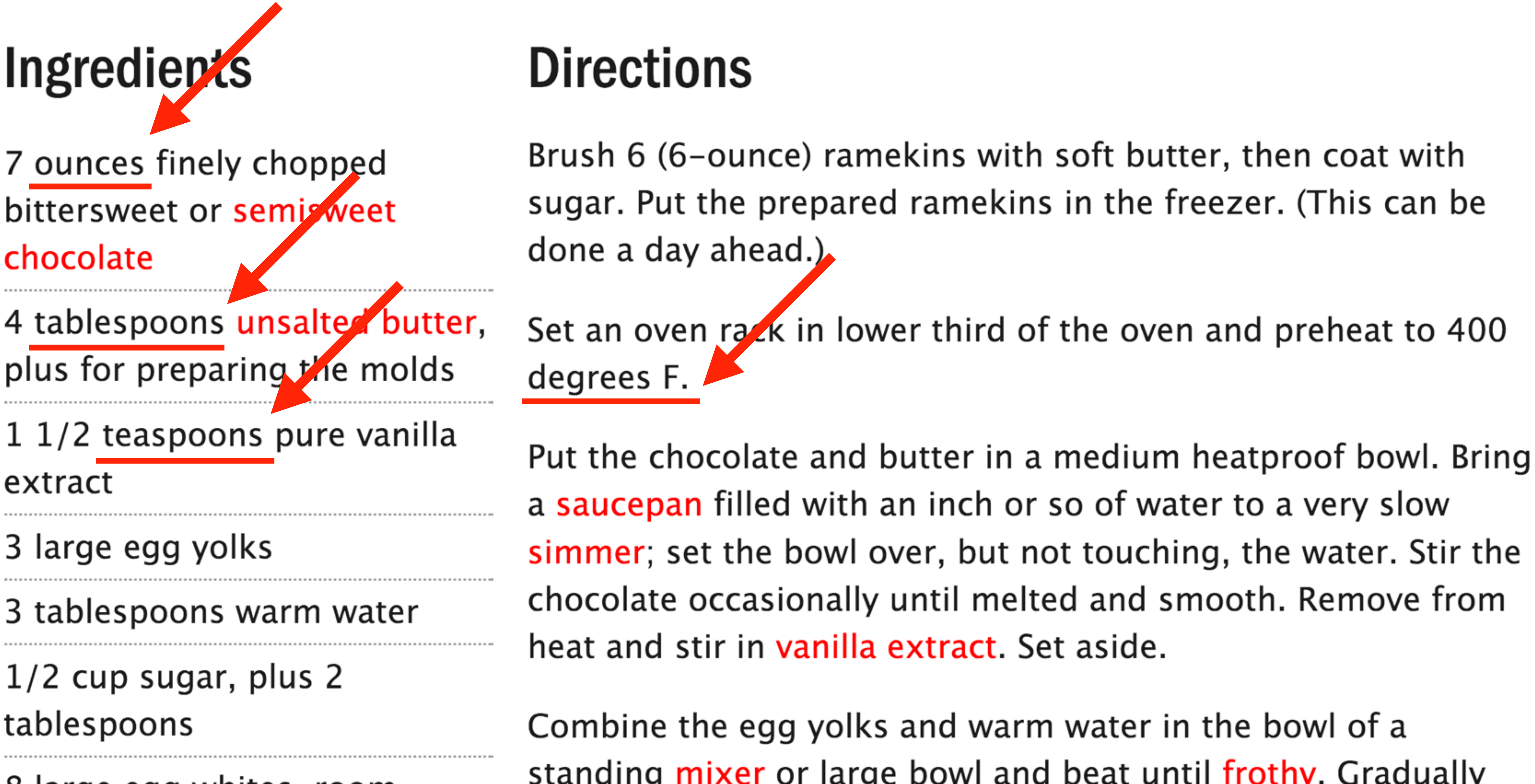
Brush 6 (6-ounce) ramekins with soft butter, then coat with sugar. Put the prepared ramekins in the freezer. (This can be done a day ahead.)

Set an oven rack in lower third of the oven and preheat to 400 degrees F.

Put the chocolate and butter in a medium heatproof bowl. Bring a saucepan filled with an inch or so of water to a very slow simmer; set the bowl over, but not touching, the water. Stir the chocolate occasionally until melted and smooth. Remove from heat and stir in vanilla extract. Set aside.

Combine the egg yolks and warm water in the bowl of a standing mixer or large bowl and beat until frothy. Gradually

LIKE A RECIPE – TYPES (KIND OF LIKE UNITS)



Ingredients

- 7 ounces finely chopped bittersweet or semisweet chocolate
- 4 tablespoons unsalted butter, plus for preparing the molds
- 1 1/2 teaspoons pure vanilla extract
- 3 large egg yolks
- 3 tablespoons warm water
- 1/2 cup sugar, plus 2 tablespoons
- 8 large egg whites, room

Directions

Brush 6 (6–ounce) ramekins with soft butter, then coat with sugar. Put the prepared ramekins in the freezer. (This can be done a day ahead.)

Set an oven rack in lower third of the oven and preheat to 400 degrees F.

Put the chocolate and butter in a medium heatproof bowl. Bring a saucepan filled with an inch or so of water to a very slow simmer; set the bowl over, but not touching, the water. Stir the chocolate occasionally until melted and smooth. Remove from heat and stir in vanilla extract. Set aside.

Combine the egg yolks and warm water in the bowl of a standing mixer or large bowl and beat until frothy. Gradually

LIKE A RECIPE – TYPES (KIND OF LIKE UNITS)

- This notion of “types” is *super* important.
- In a recipe, something like this doesn’t make sense:
  - “*Mix* 3 tablespoons of sugar *with* 400 degrees F.”
- Just like in code, values of specific *types* are sometimes compatible, most of the time not.
- We say that the “type” carries with it a set of “semantics” that only make sense for values of that type.
- “Semantics” deals with what it *means* to do certain operations on a type, e.g. *adding* values together.

LIKE A RECIPE – TYPES (KIND OF LIKE UNITS)

- Declaring types explicitly makes languages faster, because the iPhone “knows” how to allocate the proper memory without checking.
- Swift is special because it makes dealing with types much easier (except for Optionals, which can be tricky).
- Swift gives us the benefits of a fast language using types without the pain.

LET'S CODE!

SYNC THE REPO, COPY ALL PLAYGROUNDS TO EXERCISES/LESSON 03

OPEN ARITHMETIC.PLAYGROUND

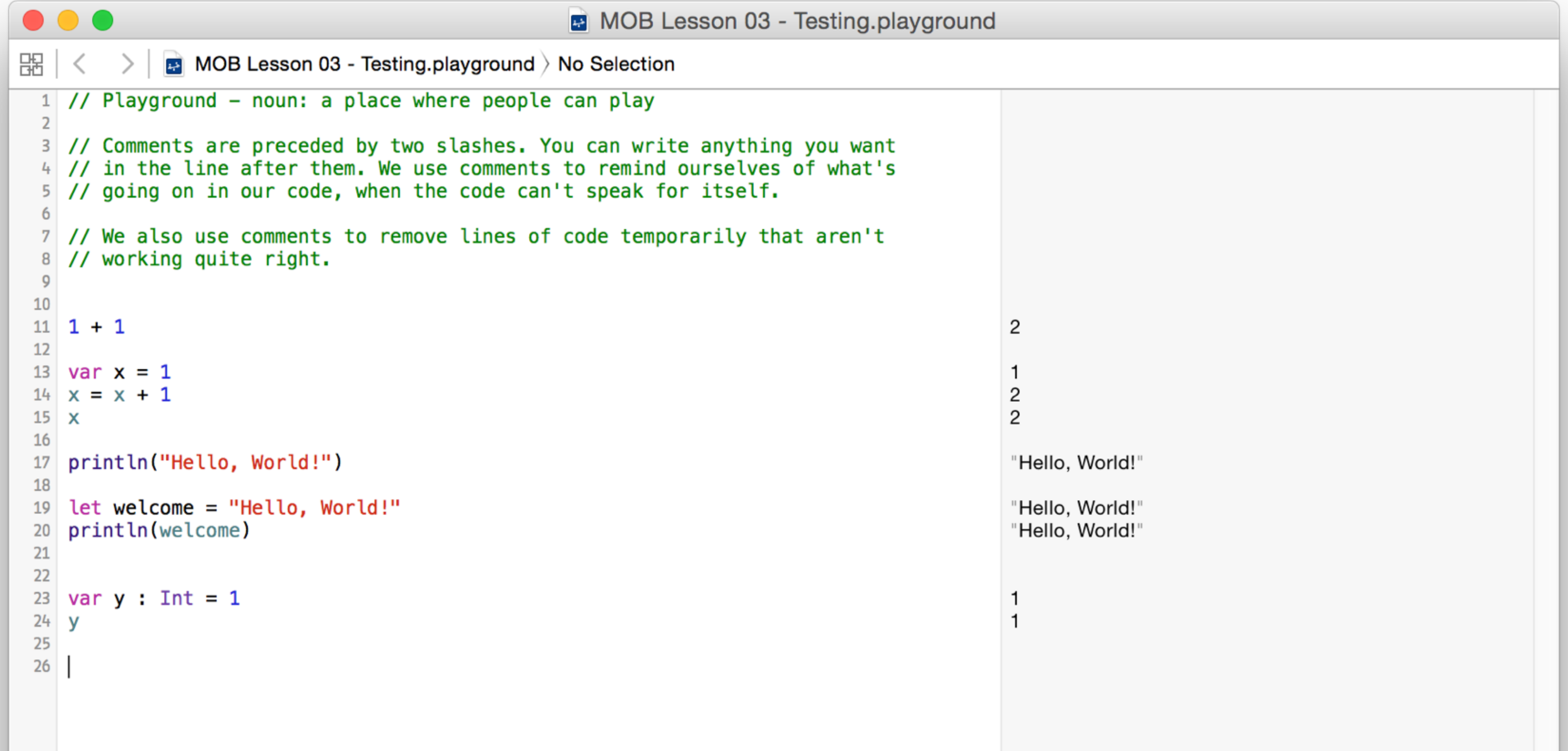
SYNTAX

- Programming (or formal) languages are similar to natural languages in that they have a written syntax that defines how characters are arranged into meaningful patterns.
- Programming languages have:
  - keywords - Words specific to the language that we can’t override (e.g. “var”).
  - operators - Symbols that take their meaning from their context (+, -, etc.).
  - comments - The ability to put plain language that Swift will ignore.
  - whitespace - Do spaces and tabs mean something? In Swift, no.
  - grouping symbols - Quotes, braces, parentheses, brackets.

VALUES AND TYPES - INTEGERS, FLOATS, DOUBLES

- Numeric types are very mathematical in nature:
  - Integers (Int): -5, -4, -3, -2, 0, 1, 2, 3, 4, 5
  - Floats (Float): 2.71828, 3.14159265, 1.0
  - Doubles (Double): Similar to Floats, just bigger. No way to distinguish them just from how the values look.
  - Note that 1 and 1.0 are *different types*
- We say that “3.14” is “of type Float.”
- Semantics: mimic arithmetic (+, -, etc.) and comparison (>, <, etc.).

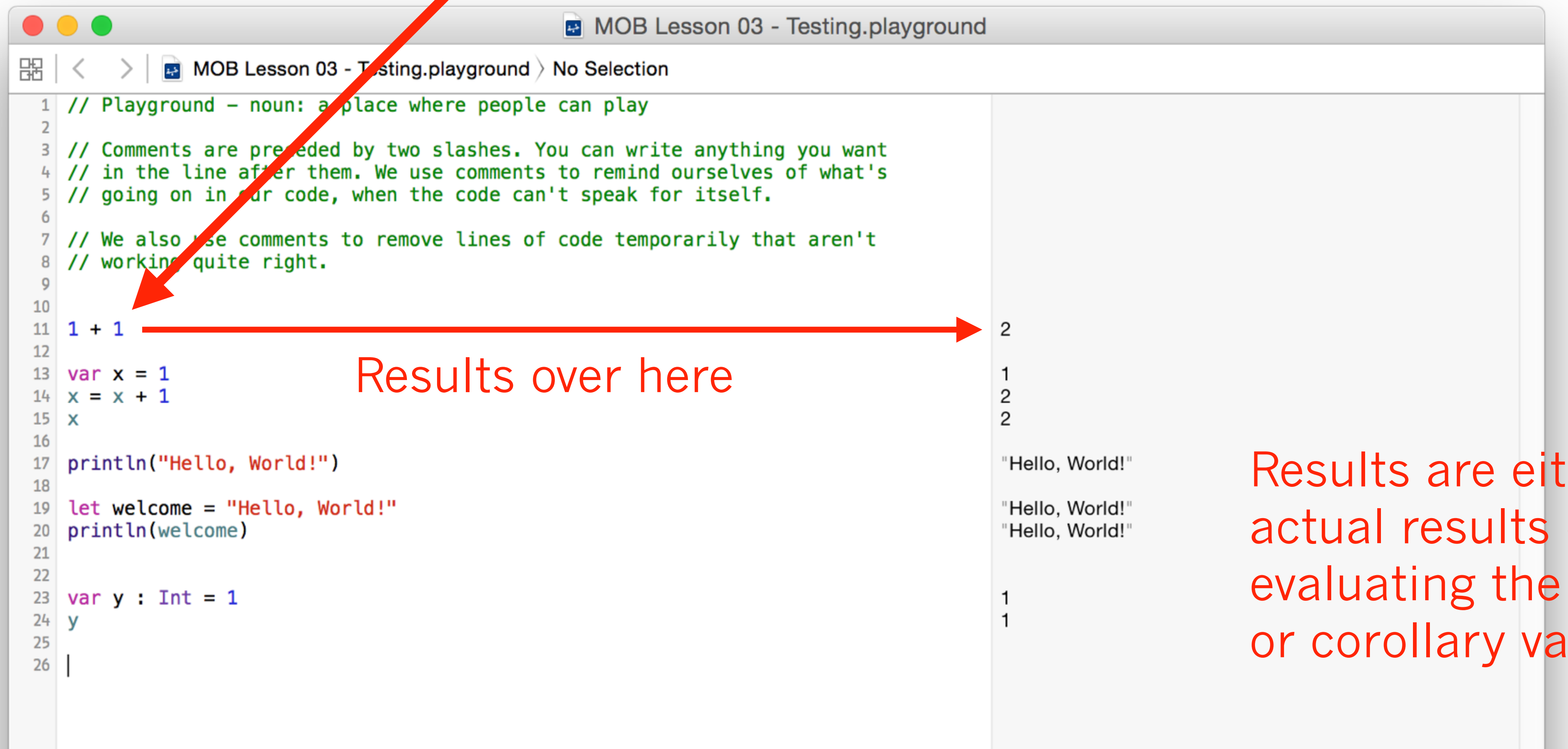
PLAYGROUNDS


MOB Lesson 03 - Testing.playground
MOB Lesson 03 - Testing.playground 〉 No Selection

```
// Playground - noun: a place where people can play

// Comments are preceded by two slashes. You can write anything you want
// in the line after them. We use comments to remind ourselves of what's
// going on in our code, when the code can't speak for itself.

// We also use comments to remove lines of code temporarily that aren't
// working quite right.


1 + 1                                                                       2

var x = 1                                                                   1
x = x + 1                                                                   2
x                                                                           2

println("Hello, World!")                                                    "Hello, World!"

let welcome = "Hello, World!"                                               "Hello, World!"
println(welcome)                                                            "Hello, World!"


var y : Int = 1                                                             1
y                                                                           1

```

PLAYGROUNDS

Lines of code here


MOB Lesson 03 - Testing.playground
MOB Lesson 03 - Testing.playground 〉 No Selection

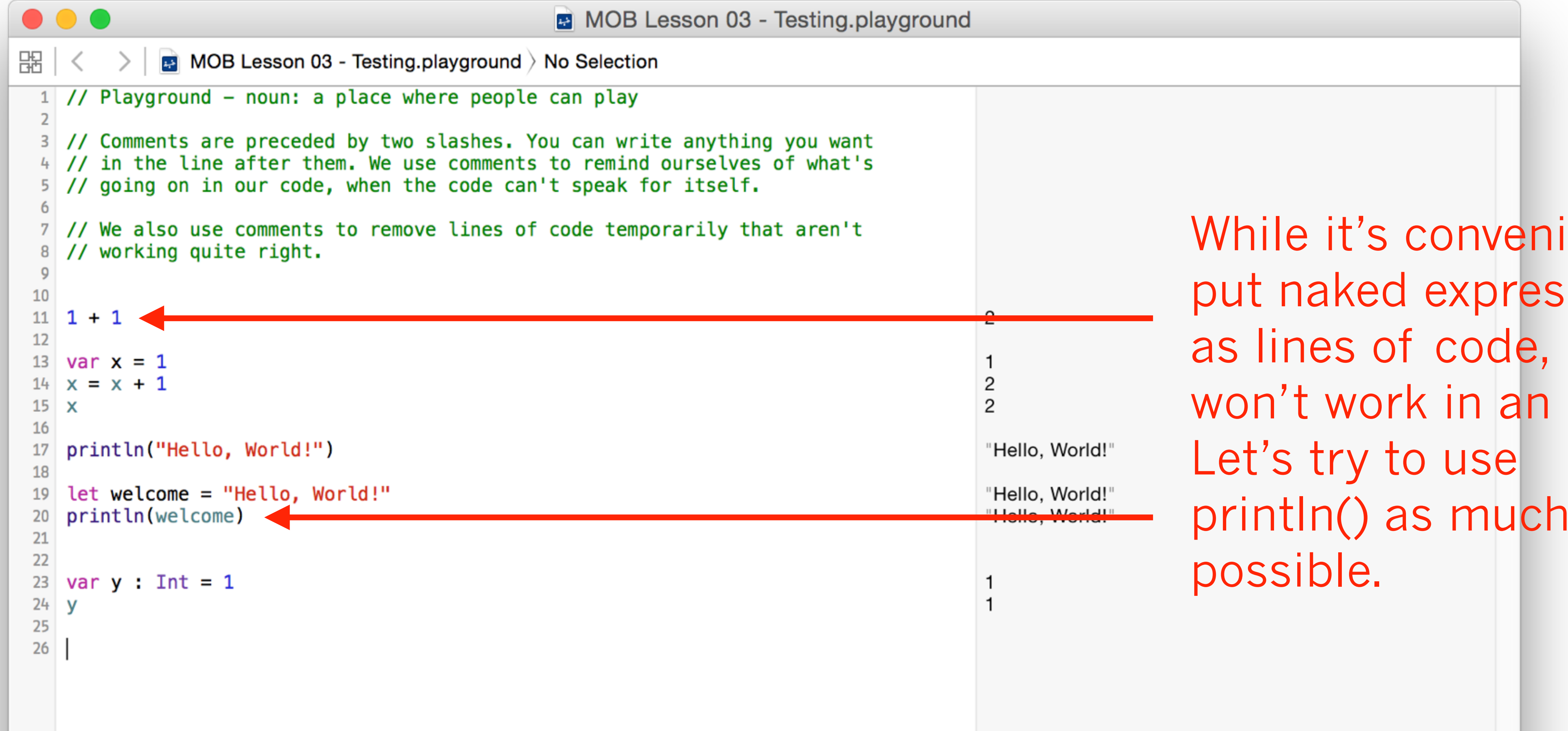
```
// Playground - noun: a place where people can play

// Comments are preceded by two slashes. You can write anything you want
// in the line after them. We use comments to remind ourselves of what's
// going on in our code, when the code can't speak for itself.

// We also use comments to remove lines of code temporarily that aren't
// working quite right.


1 + 1

var x = 1
x = x + 1
x

println("Hello, World!")

let welcome = "Hello, World!"
println(welcome)


var y : Int = 1
y

```

```
2

1
2
2

"Hello, World!"

"Hello, World!"
"Hello, World!"


1
1
```

Results over here

Results are either actual results of evaluating the code or corollary values.

PLAYGROUNDS


MOB Lesson 03 - Testing.playground
MOB Lesson 03 - Testing.playground 〉 No Selection

```swift
// Playground - noun: a place where people can play

// Comments are preceded by two slashes. You can write anything you want
// in the line after them. We use comments to remind ourselves of what's
// going on in our code, when the code can't speak for itself.

// We also use comments to remove lines of code temporarily that aren't
// working quite right.


1 + 1

var x = 1
x = x + 1
x

println("Hello, World!")

let welcome = "Hello, World!"
println(welcome)


var y : Int = 1
y

```

```
2

1
2
2

"Hello, World!"

"Hello, World!"
"Hello, World!"


1
1
```

While it’s convenient to put naked expressions as lines of code, that won’t work in an app. Let’s try to use println() as much as possible.

OPEN STRINGS.PLAYGROUND

VALUES AND TYPES – STRINGS

- Strings represent a sequence of characters.
- Delineated by double-quotes
- Examples
    - `"Hello, World!"`
    - `"1.0"`
- Note that 1.0 and "1.0" are *different types*. The former is a Float, the latter, a String. They have different semantics, and thus don’t play well nicely together.

OPEN BOOLEANS.PLAYGROUND

VALUES AND TYPES – BOOLEANS (BOOL)

- Booleans (type Bool) represent “trueness” and “falseness.”
- A Bool is a type used for digital logical reasoning.
- The only two possible values are:
    - `true`
    - `false`
- We can use special language constructs in tandem with Bools to “control the flow” of the code that drives our apps. (Later in this lesson.)

VALUES AND TYPES – BOOLEANS (BOOL)

- Numeric types have what we call binary comparison operators that take two numbers and become a Bool.
- Similar to arithmetic operators (+, -, *, /, %) which take two numbers and produce a number.
- Examples:
    - Less than: 3 < 1, less than or equal to: 3 <= 1
    - Greater than: 3 > 1, greater than or equal to: 3 >= 1
    - Equality: 3 == 3
    - Inequality: 3 != 2

VALUES AND TYPES – BOOLEANS (BOOL)

- Boolean operators take one or two Bools and produce one Bool.
- They are:
  - AND: true && true, true && false, false && false
  - OR: true || true, true || false, false || false
  - NOT: !true
- You can use parentheses to dictate the order of operations
  - (1 < 3) && (3 < 5)

OPEN VARIABLES.PLAYGROUND

VARIABLES AND CONSTANTS

- Variables
  - Symbols that represent *changeable* state *of a particular type.*
  - Contains a value of that type, and that *value* can change, i.e. *mutable.*
- Constants
  - Symbols that represent *unchangeable* state *of a particular type.*
  - Contains a value of that type, but the *value* never changes, i.e. *immutable.*
- *Neither variables nor constants can have their type changed.*

VARIABLES AND CONSTANTS

- Variables are “declared” by using the keyword “var”.
- *Keywords* are symbols in the language that are reserved for use by Swift. We can’t repurpose them for our own usage.
- Variables are “initialized” when they are given their first value using =.
- The basic templates for declaring (and initializing) variables is:
    - var [symbol] = [value]
    - var [symbol] : [type] = [value of type]
    - var [symbol] : [type]
- Once a variable is declared, that symbol is available for every subsequent line.

VARIABLES AND CONSTANTS

- Examples of declaring and initializing variables
  - `var x = 1`
  - `var y : Double = 1.0`
  - `var isEasy : Bool = true`
- Declaring and initializing constants
  - `let c = 299792458`
  - `let c : Double = 299792458`
  - `let canChange : Bool = false`

VARIABLES AND CONSTANTS – ASSIGNING VALUES

Examples of assigning new values to a variable already declared:

```
// Change the value of x.
var x = 1
// Do stuff with x.
x = 2
// Do more stuff with x.
x = 3
```

Note that assignments always work right-to-left. We compute the value to the right of = and assign it to x.

VARIABLES AND CONSTANTS – ASSIGNING VALUES

Examples of assigning new values to a variable already declared:

```
// Increment the variable by one.
var x = 1
x = x + 1
x += 1
x++
++x
```

CONTROL FLOW

CONTROL.PLAYGROUND

CONTROL FLOW

- Programs are executed one line at a time, but it's not useful to execute all lines of code all of the time.
- Conditional statements leverage Boolean expressions to begin to define the logic of our apps. We can execute some code under certain conditions, and other code under other conditions.
- We can start to reason like this:
  - e.g. "If the temperature is less than or equal to 32 degrees, show a freezing icon, otherwise, show water drop icon."
- Also, we can start to leverage a computer's automation abilities by using loops.
  - e.g. "Keep executing this code as long as the temperature is less than 32."

CONTROL FLOW – CONDITIONALS

Conditional statements, or “if-else” statements, look like this:

```
if temp <= 32 {
    // This “block” is executed if the condition is true.
    // Show a freezing icon.
} else {
    // And this “block” if false.
    // Show a water drop icon.
}
```

CONTROL FLOW – CONDITIONALS

Conditional statements can contain multiple blocks or clauses, using “else if”:

```
if temp <= 32 {
    // Show a freezing icon.
} else if temp >= 212 {
    // Show a boiling water icon.
} else {
    // Show a water drop icon.
}
```

CONTROL FLOW – WHILE LOOPS

The simplest kind of loop, while loops execute a block of code repeatedly as long a given condition is true.

```
var sum = 0
while sum < 50 {
    sum += 10
}
println(sum)
```

CONTROL FLOW – FOR LOOPS

Strangely named, “for-loops” use conditionals to continue executing code given a conditional and a variable that is used for counting.

```
for (var temp=0; temp<=32; temp++) {
    // Do something here.
}
```

CONTROL FLOW – FOR LOOPS

```
for (var temp=0; temp<=32; temp++) {
    // Do something here.
}
```

1. The loop declares and initializes a variable (temp),
2. checks the conditional, and if it’s true,
3. executes the block of code within the braces, then
4. calls the incrementing expression (temp++)
5. checks the conditional again, etc.

CONTROL FLOW – CONTROL TRANSFER – BREAK

```
let toCheck = 289
for (var i=2; i<toCheck; i++) {
    println(i)
    if toCheck % i == 0 {
        println(“composite!”)
        break
    }
}
```

The “break” statement aborts from the for loop.

Advanced students: make this more efficient. Write as a while loop.

CONTROL FLOW – CONTROL TRANSFER – CONTINUE

```
let toCheck = 289
for (var i=2; i<toCheck; i++) {
    if i % 2 == 0 { continue }
    if toCheck % i == 0 {
        println(“composite!”)
        break
    }
}
```

The “continue” statement skips everything after it in the block, but continues executing the loop.

OPEN OPTIONALS.PLAYGROUND

OPTIONALS AND NIL

- `nil`
    - A value that represents no value.
- Optional - a type that represents `nil` or a value of another specified type
- Syntax:
    - `var [symbol] : [type]?`
- Example
    - `var name : String?   // initialized as nil`
    - `var name : String? = “Toshi”`

OPTIONALS AND NIL

- Why use Optionals?
  - Sometimes we need a variable before we get a chance to give it a real value.
  - e.g. Imagine a web request that takes some time. We need a place to put the response to that query, but we won’t know what the response is until the request is done.

CODE ALONG EXERCISE IN PAIRS

GETTING STARTED

EXERCISE

KEY OBJECTIVE(S)

Demonstrate basic data types, variables, constants, optionals, and type annotations.

TIMING

30 min	1. Code with partner
5 min	2. Debrief

DELIVERABLE

To the best of your ability, complete the provided playground file. If you hit a question you don’t feel comfortable with, ask an instructor.
Bonus 1: Research and use the ternary operator
Bonus 2: Research and use the nil coalescing operator

RECAP

- Variables: Changeable state
- Constants: Unchangeable state
- Type: What a variable/constant is, e.g. String, Int, Bool, Float
- nil: Nothing, the absence of a value
- Optional: A kind of type that can be nil

RECAP

- In Swift, all variables have a type (e.g. String, Int).
    - You cannot change the type of a variable.
- Some things are constant, i.e. they cannot be changed.
- If a variable can be nil, you must define it as Optional.

OPTIONALS

- If something in Swift **can be** nil (i.e. it is Optional), that must be part of its type:
    - Syntax:
        - var i = 1 // Int
        - var i: Int? = 1 // Optional Int, we can set i to nil some time later.
- If a variable in Swift is not Optional, we must set a value for it immediately.

SYNTAX REVIEW

- **var** *variableName***:** *Type* **=** *value*
- **var** *optionalVariableName***:** Type**? =** *valueOrNil*
- Examples:
    - var name: String = “rudd” // Creates a changeable variable, of type String, set to the value “rudd”
    - var name = “rudd” // Same as above, as Swift is smart about types
    - let age = 30 // age is constant, of type Int
    - var age: Int? // Unless our variables are optional, we MUST assign them.

RECAP

- Swift gives us a number of ways to control what code gets run, and when
  - if/else if/else: Used when we only want to run code under certain circumstances
  - for/while: Used when we want to run the same block of code multiple times, e.g. for each element in a list we want to perform an action
  - if let: Used when we want to turn optional variables into non-optional variables, if they exist. This process is called ‘unwrapping’

SYNTAX REVIEW

- **if** *statement* { *code* } // Code runs if statement evaluates to true
- **if** *statement* { *code* } **else** { *moreCode* } // Code runs if statement evaluates to true, moreCode runs if statement is false
- **if** *statement* { *code* } **else if** *statement2* { *moreCode* } // Code runs if statement evaluates to true, moreCode runs if statement2 is true
  - You can stack as many if else blocks as you want.
- **if let** *name* **=** *optional* { *code* } // code runs and has access to a non-optional version of *optional,* called *name*, only if *optional* exists
- for, while // Loops

REVIEW

CLASS REVIEW

- How often do students submit assessments?
- What is a typed language? Is Swift typed?
- What is the difference between a compiled and scripted language? Which one is Swift?