GENERAL ASSEMBLY

MOBILE DEVELOPMENT

INTRO TO CONTROL FLOW

William Martin
Head of Product, Floored

Angel X. Moreno
EIR, Developer

LEARNING OBJECTIVES

- Review variables, types, values.
- Utilize “control flow” to make simple programs.
- Apply “Optionals” and explain when and how to use them.
- Identify functions and implement best practices
- Be able to call and define functions that take parameters
- Be able to use the returned value from a function
- Understand what returning from a function does

REVIEW LESSON 3

INTRO TO CONTROL FLOW

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.

CONTROL FLOW

- 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.

OPTIONALS - UNWRAPPING

- Optionals have two somewhat incompatible states:
  - nil, representing no value
  - has a value of a particular type
- In order to get to an Optional's value (if it's not nil), we have to "unwrap" by adding an ! right after the variable:
  - `var name : String?`
  - `name = "Toshi"`
  - `println("My pup's name is \(name!).")`

OPTIONALS - UNWRAPPING

- However, there's a problem with unwrapping. You can't unwrap an Optional if it's nil. In that case, the syntax `name!` would cause an error.
- How do we deal with this? This syntax helps us distinguish between the nil and value-holding cases, and also unwraps the value if it's available (i.e. not nil):

```
if let _name = name {
    println("The pup's name is \(_name).")
} else {
    println("I don't know the pup's name…")
}
```

REVIEW