Control Flow

Use `if` and `switch` to make conditionals, and use `for-in`, `while`, and `repeat-while` to make loops. Parentheses around the condition or loop variable are optional. Braces around the body are required.

```
let individualScores = [75, 43, 103, 87, 12]
var teamScore = 0
for score in individualScores {
    if score > 50 {
        teamScore += 3
    } else {
        teamScore += 1
    }
}
print(teamScore)
// Prints "11"
```

In an `if` statement, the conditional must be a Boolean expression—this means that code such as `if score { ... }` is an error, not an implicit comparison to zero.

You can use `if` and `let` together to work with values that might be missing. These values are represented as optionals. An optional value either contains a value or contains `nil` to indicate that a value is missing. Write a question mark (`?`) after the type of a value to mark the value as optional.

```
var optionalString: String? = "Hello"
print(optionalString == nil)
// Prints "false"

var optionalName: String? = "John Appleseed"
var greeting = "Hello!"
if let name = optionalName {
    greeting = "Hello, \(name)"
} else {
    greeting = "Hello, is anyone here?"
}
```

EXPERIMENT

Change `optionalName` to `nil`. What greeting do you get? Add an `else` clause that sets a different greeting if `optionalName` is `nil`.

If the optional value is `nil`, the conditional is `false` and the code in braces is skipped. Otherwise, the optional value is unwrapped and assigned to the constant after `let`, which makes the unwrapped value available inside the block of code.

Another way to handle optional values is to provide a default value using the `??` operator. If the optional value is missing, the default value is used instead.

```
let nickname: String? = nil
let fullName: String = "John Appleseed"
let informalGreeting = "Hi \(nickname ?? fullName)"
```

Switches support any kind of data and a wide variety of comparison operations—they aren't limited to integers and tests for equality.

```
let vegetable = "red pepper"
switch vegetable {
case "celery":
    print("Add some raisins and make ants on a log.")
case "cucumber", "watercress":
    print("That would make a good tea sandwich.")
case let x where x.hasSuffix("pepper"):
    print("Is it a spicy \(x)?")
default:
    print("Everything tastes good in soup.")
}
// Prints "Is it a spicy red pepper?"
```

> EXPERIMENT
>
> Try removing the default case. What error do you get?

→ error: switch must be exhaustive

Notice how `let` can be used in a pattern to assign the value that matched the pattern to a constant.

After executing the code inside the switch case that matched, the program exits from the switch statement. Execution doesn't continue to the next case, so you don't need to explicitly break out of the switch at the end of each case's code.

You use `for-in` to iterate over items in a dictionary by providing a pair of names to use for each key-value pair. Dictionaries are an unordered collection, so their keys and values are iterated over in an arbitrary order.

```
let interestingNumbers = [
    "Prime": [2, 3, 5, 7, 11, 13],
    "Fibonacci": [1, 1, 2, 3, 5, 8],
    "Square": [1, 4, 9, 16, 25],
]
var largest = 0
for (_, numbers) in interestingNumbers {
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}
print(largest)
// Prints "25"
```

EXPERIMENT

Replace the `_` with a variable name, and keep track of which kind of number was the largest.

Use `while` to repeat a block of code until a condition changes. The condition of a loop can be at the end instead, ensuring that the loop is run at least once.

```
var n = 2
while n < 100 {
    n *= 2
}
print(n)
// Prints "128"

var m = 2
repeat {
    m *= 2
} while m < 100
print(m)
// Prints "128"
```

You can keep an index in a loop by using `..<` to make a range of indexes.

```
var total = 0
for i in 0..<4 {
    total += i
}
print(total)
// Prints "6"
```

Use `..<` to make a range that omits its upper value, and use `...` to make a range that includes both values.