# Swift/iOS Tutorial for 6.808

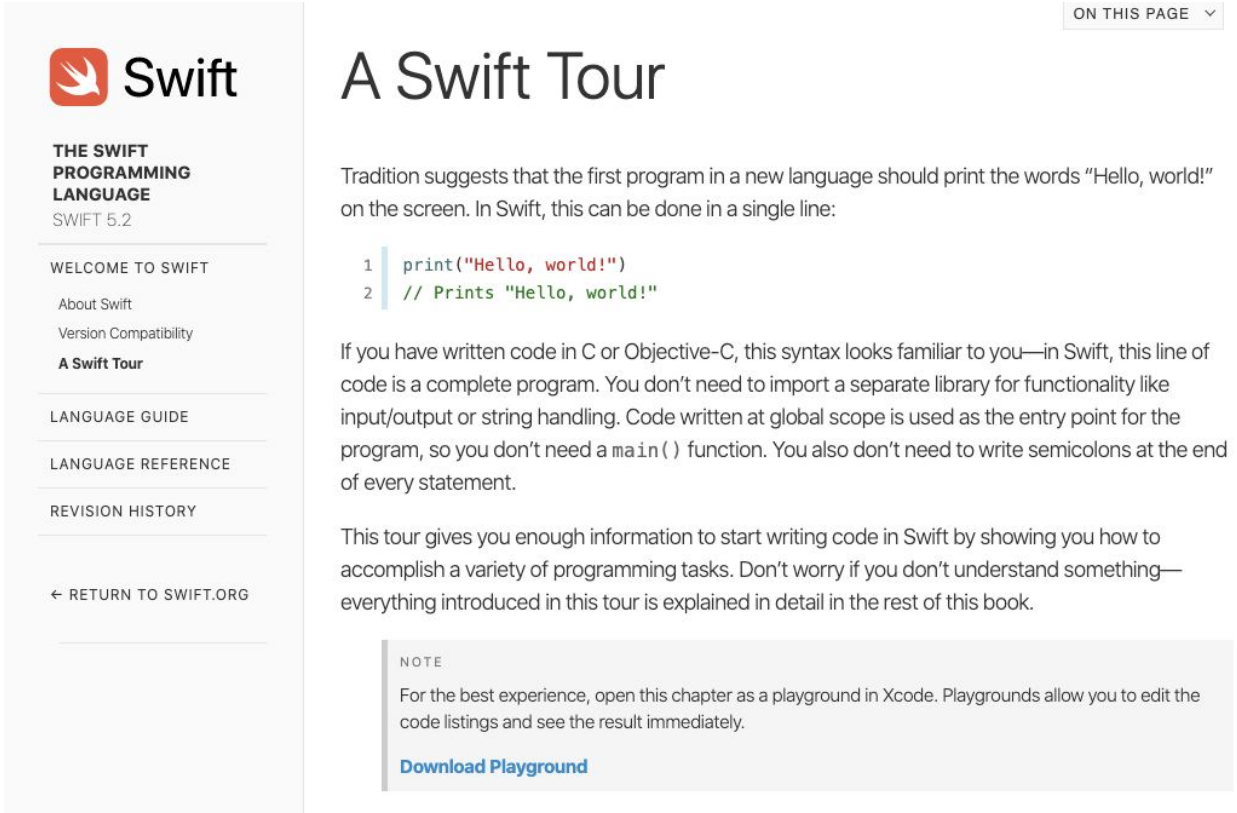
Feb 19, 2021

# This tutorial is based on Apple's Official Guide

I will point out important features that may be used in 6.808 labs.

Links

- A Swift Tour
- The Basics



# Constants, Variables, Types

Constants (`let`) and Variables (`var`)

```
var myVariable = 42
myVariable = 50
let myConstant = 42
```

Type Safety

- Auto type inference
- Annotate with a colon if needed
- Static type checking
- Explicit type conversion required
  - e.g. Int <-> Double

```
let implicitInteger = 70
let implicitDouble = 70.0
let explicitDouble: Double = 70
```

# String Interpolation \( . . . )

- Convert values to string
- Allow calculations

```
let apples = 3
let oranges = 5
let appleSummary = "I have \(apples) apples."
let fruitSummary = "I have \(apples + oranges) pieces of fruit."
```

# Printing

- `print` vs. `NSLog`
  - NSLog adds a timestamp and identifier (slower)
  - NSLog is used in Lab 0 video tutorials

Xcode console

```
print("Hello, world!")   ⟶   Hello, world!

NSLog("Hello, world!")   ⟶   2020-02-07 09:22:39.947443-0500 Lab0[87933:5960518] Hello, world!
```

# Arrays, Dictionaries

### Arrays

```
var shoppingList = ["catfish", "water", "tulips"]
shoppingList[1] = "bottle of water"
shoppingList.append("blue paint")
```

### Methods

- `lst.count`
- `lst.enumerated()`
- `lst.isEmpty`

Python

`len()`

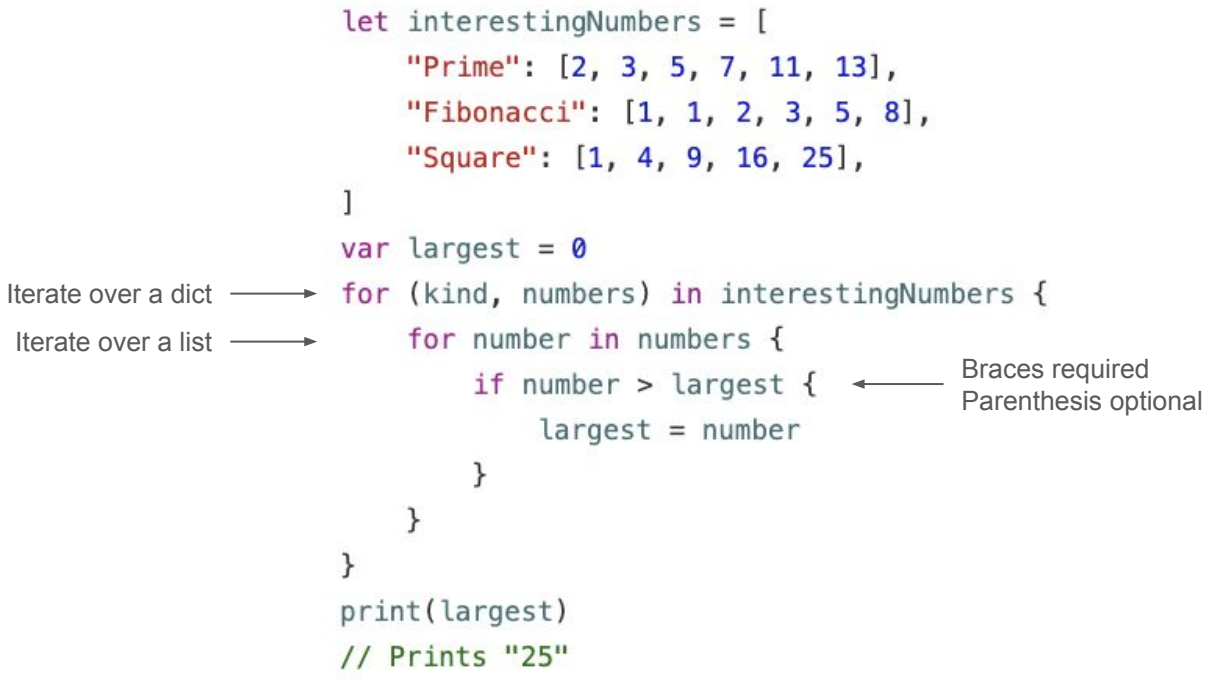
`enumerate()`

**Dictionaries** use `[k:v]`, not `{k:v}`.

```
var occupations = [
    "Malcolm": "Captain",
    "Kaylee": "Mechanic",
]
occupations["Jayne"] = "Public Relations"
```

# Control Flow

- Conditionals: `if`, `switch`
- Loops: `for-in`, `while`, `repeat-while`




```
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 (kind, numbers) in interestingNumbers {
    for number in numbers {
        if number > largest {
            largest = number
        }
    }
}
print(largest)
// Prints "25"
```

# Range Operators

- Closed range operator `a...b`
- Half-open range operator `a..<b`
- One-sided ranges `...a`, `a...`, `..<a`

```
let names = ["Anna", "Alex", "Brian", "Jack"]
let count = names.count
for i in 0..<count {
    print("Person \(i + 1) is called \(names[i])")
}
// Person 1 is called Anna
// Person 2 is called Alex
// Person 3 is called Brian
// Person 4 is called Jack
```

```
for name in names[2...] {
    print(name)
}
// Brian
// Jack

for name in names[...2] {
    print(name)
}
// Anna
// Alex
// Brian
```

# Optionals

- Value or `nil`

Type followed by ?

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

- Optional binding: `if/while + let` pattern

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

`name`'s type is `String`

Can add the `else` case for `nil`

# Forced unwrapping, Downcasting

- Forced unwrapping: a !
  - “I know that this optional definitely has a value; please use it.”

```
let possibleNumber = "123"
let convertedNumber = Int(possibleNumber)
// convertedNumber is inferred to be of type "Int?", or "optional Int"

if convertedNumber != nil {
    print("convertedNumber has an integer value of \(convertedNumber!).")
}
// Prints "convertedNumber has an integer value of 123."
```

- Downcasting: type cast operator
  - `as?` Conditional: return an optional value
  - `as!` Forced: trigger a runtime error if downcast to an incorrect class type

# Functions

- `func`; use `->` followed by return type

```
func greet(person: String, day: String) -> String {
    return "Hello \(person), today is \(day)."
}
greet(person: "Bob", day: "Tuesday")
```

- Custom/no argument labels

# Extra: Empty array/dictionary

- Initializer

```
let emptyArray = [String]()
let emptyDictionary = [String: Float]()
```

- If type can be inferred, e.g. set value, pass an argument to a function

```
shoppingList = []
occupations = [:]
```