Functions

- Self-contained chunks of code to perform a specific task
- Function name is called in order to perform the task
- A function call (outside of a class) looks like:

```func function-name(argument) -> return-type { /* code here */ }```
Hello World Example

```swift
func helloWorld() -> String {
    return "Hello World"
}

Call using helloWorld()

What does this do?
Hello Example with Parameters

```swift
func sayHello(personName: String) -> String {
    return "Hello, " + personName
}
```

- How do we pass in “Dave” as `personName`?
- What does this function then return?
Parameters

- Function call arguments correspond to function parameters
  - `personName` is the parameter
  - “Dave” is the argument

- Argument type must match parameter type
- Order of arguments must match order of parameters
Return Values

Functions do not have to include return values (will return `void` if none are specified)

If there’s no return value, drop `-> return-type`

```swift
func sayGoodbye() { print("Goodbye!") }
```

Functions can return multiple values (tuples)

```swift
func giveMeThings() -> (Int, Float, String) { return(5, 300.0, "Foo") }
```
External Names

- Provide a convenient way to clarify argument names
  
  ```swift
  func callMe(extName name: String) {
    print("Call me \(name).")
  }
  ```

- If provided, must be used when calling the function
  
  ```swift
  callMe(extName:"Tim")
  ```
Named Types

- Data type that can be named when defined
- Includes classes, structures, enumerations, protocols, and primitives
- Examples:
  - Int (Built-in)
  - myClass (user-defined)
- Behavior is extendable
Compound Types

- Unnamed data type
- Two compound types:
  - Tuple types
  - Function types
- Can contain named types or other compound types
Tuple Type

- List of zero or more types
- Comma-separated, enclosed in parenthesis

```swift
var origin:(Int, Int) = (0, 0)
```

- Tuple elements can be named

```swift
var origin:(x: Int, y: Int) = (0, 0)
```

or

```swift
var origin = (x: 0, y: 0)
```
Accessing Tuples

- Tuples have fixed size once created
- Tuples can contain multiple types
- Elements accessible (and modifiable) with dot

```swift
var origin:(Int, Int) = (0, 0)
print(origin.0) //prints 0
origin.0 = 20
print(origin.0) //prints 20
```
Zero and One Tuples

- Special cases for tuples
  - A tuple with zero types is () or void
  - A tuple with one type is that type
    - (Int) has type Int
Function Types

- Represents type of function, method, or closure
- `parameter-type -> return-type`
- Parameter and return type can be tuples
- Can take multiple parameters and multiple return values
- What is the type of this function?

```swift
func sum(x: Int, y: Int) -> (output: Int)
```
Three primary collection types in Swift:

- Arrays, sets, and dictionaries

Can be mutable or immutable if `var` or `let` respectively.

<table>
<thead>
<tr>
<th>Array</th>
<th>Set</th>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indexes</td>
<td>Values</td>
<td>Keys</td>
</tr>
<tr>
<td>0</td>
<td>Six Eggs</td>
<td>YYZ</td>
</tr>
<tr>
<td>1</td>
<td>Milk</td>
<td>DUB</td>
</tr>
<tr>
<td>2</td>
<td>Flour</td>
<td>LHR</td>
</tr>
<tr>
<td>3</td>
<td>Baking Powder</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Bananas</td>
<td></td>
</tr>
</tbody>
</table>
Arrays

- Stores values of same type in ordered list
- Create an empty array
  ```swift
  var intArray = [Int] ()
  ```
- Create an array with default values
  ```swift
  var defaultArray = [Int](repeating: 0, count: 3)
  ```
- Create an array with initial values (array literal)
  ```swift
  var someArray:[Int] = [0, 1, 2, 3]
  ```
Modifying Arrays

- Concatenate arrays with addition operator

  ```
  let newArray = defaultArray + someArray
  ```

- Append items with `append()` or `+=`

- Access items with subscript syntax `[index]`

- `count` returns number of items

- And many other properties/functions that you will discover on your own!
Iterating over an Array

- **for-in** loops access all items within an array

```swift
for number in defaultArray
    { print(number) }
```

- **forEach** method accesses each value in the array

```swift
defaultArray.forEach { value in
    print("Value is \(value)")
}
```
Iterating over an Array

Enumerate() method provides the item’s index

```python
for (index, value) in defaultArray.enumerated()
    print("Index \(index) is \(value)")
```
Sets

- Store distinct values of the same type with no defined order
- Item can appear only once in a set
- Allow for set operations:
  - Union, Intersect, Subtract, ExclusiveOr
- Will not be discussing these in great detail…
Dictionaries

- Store associations between keys of same type and values of same type with no defined ordering

- Create an empty dictionary

  ```
  var namesOfIntegers = [Int: String]() 
  ```

- Create a dictionary with initial values (dictionary literal)

  ```
  var airports:[String: String] = ["YYZ": "Toronto", "DUB": "Dublin"]
  ```
Modifying Dictionaries

- Add new item with new key of appropriate type
  
  ```
  airports["LHR"] = "London"
  ```

- Change value associated with a key
  
  ```
  airports["LHR"] = "London Heathrow"
  ```

- Return and remove a key-value pair or return `nil`
  
  ```
  airports.removeValueForKey("DUB")
  ```
Iterating over a Dictionary

- for-in loops can use key-value pairs as tuples

```python
for (code, name) in airports { print("\n(code): \(name)") }
```

- Or access key or value properties

```python
for code in airports.keys
{ print("Code: \(code)") }
```