GENERAL ASSEMBLY

SWIFT FUNCTIONS (CLOSURES)

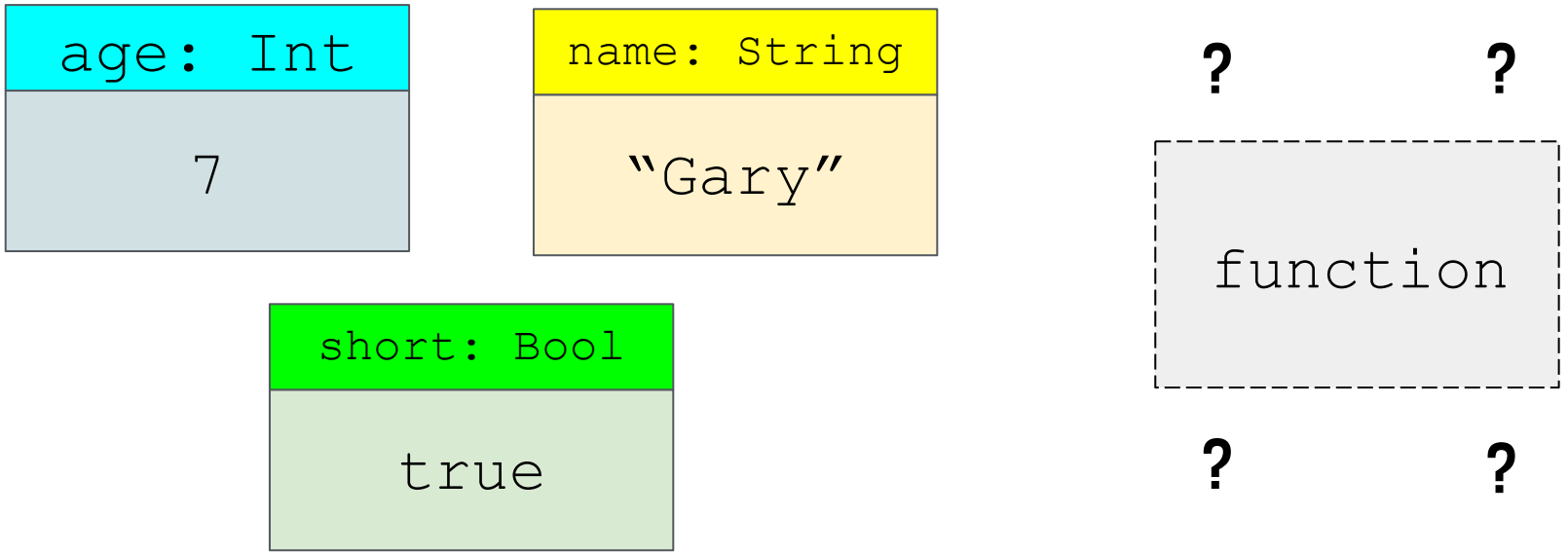
Zeke Abuhoff

Lead iOS Instructor, General Assembly

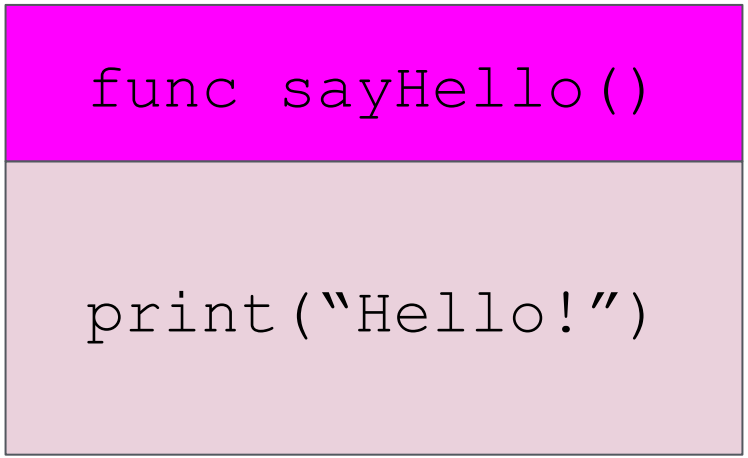
LEARNING OBJECTIVES

+ Define a closure
+ Declare a closure variable
+ Call functions that accept closures as parameters
+ Write functions that accept closures as parameters

WHAT IS A FUNCTION?



FUNCTIONS ARE OBJECTS!



CLOSURES

"Closures are self-contained blocks of functionality that can be passed around and used in your code." -Apple

CLOSURES

> “Closures are self-contained blocks of functionality that can be passed around and used in your code.” -Apple

All functions are closures. When we write them, we’re just defining a closure object.

CLOSURES

> "Closures are self-contained blocks of functionality that can be passed around and used in your code." -Apple

All functions are closures. When we write them, we're just defining a closure object.

Like an object of any other type, closures can be instantiated and passed as a parameter.

Unlike other objects, closures can be run.

CLOSURES

```
var closureName: (ParameterTypes) -> (ReturnType)
```

```
var errand: () -> (Bool)

let printHello = {
   print(“Hello!”)
}
```

CLOSURES

```
var closureName: (ParameterTypes) -> (ReturnType)
```

Practice

1) Declare a closure variable that takes an Int and returns a String.
2) Define a closure constant that takes a string and prints it with an exclamation point on the end.

PASSING CLOSURES

```
funcName({ (parameters) -> ReturnType in return returnValue})
```

```
someFunc(myClosure)


someOtherFunc({ (myString) -> Bool in
    print(myString)
    return true
})
```

PASSING CLOSURES

```
funcName({ (parameters) -> ReturnType in return returnValue})
```

Practice

1) Copy the provided function that takes a closure.
2) Call the provided function once, passing it a closure that prints the passed string parameter twice.
3) Define a constant with the return value of the function call you wrote for the previous task.

TAKING CLOSURES

```
func funcName(closure: (ParameterTypes) -> (ReturnType))
```

```
func doThreeTimes(doable: () -> ()) {
    for 1...3 {
        doable()
    }
}
```

TAKING CLOSURES

```
func funcName(closure: (ParameterTypes) -> (ReturnType))
```

Practice

1) Write a function that takes a closure as a parameter. The closure parameter should have no parameters or return type. The function should run the closure once, then print “Done!”
2) Write a function that takes a string and a closure as parameters. The closure parameter should take a string and return a string. The function should run the closure, passing it its own string parameter with an exclamation point added.