Reverse Engineering Swift Apps

Michael Gianarakis

Hack In The Box GSEC 2016

Trustwave® SpiderLabs®

# whoami

@mgianarakis

Director of SpiderLabs at Trustwave

SecTalks Organiser

Flat Duck Justice Warrior (FDJW)

Motivation

- Seeing more and more Swift being used in apps that we test (fan boys like me tend to adopt new Apple technology quickly)

- Google is even considering using Swift as a first class language on Android… (http://thenextweb.com/dd/2016/04/07/google-facebook-uber-swift/)

- Wanted to dive into some of the key differences with Swift and look at the challenges with respect to Swift app pen testing

- Focus is on "black box" app pen testing - for a deeper dive into Swift language RE I recommend Ryan Stortz's talk at Infiltrate (http://infiltratecon.com/archives/swift_Ryan_Stortz.pdf)

How Does Swift Affect Testing?

- Will dive into the detail in the presentation but the reality is not much in most areas, quite a bit in others?

- Most issues in iOS and OS X apps are due to poor design decisions or misconfiguration and incorrect implementation of Apple and third party frameworks and libraries.

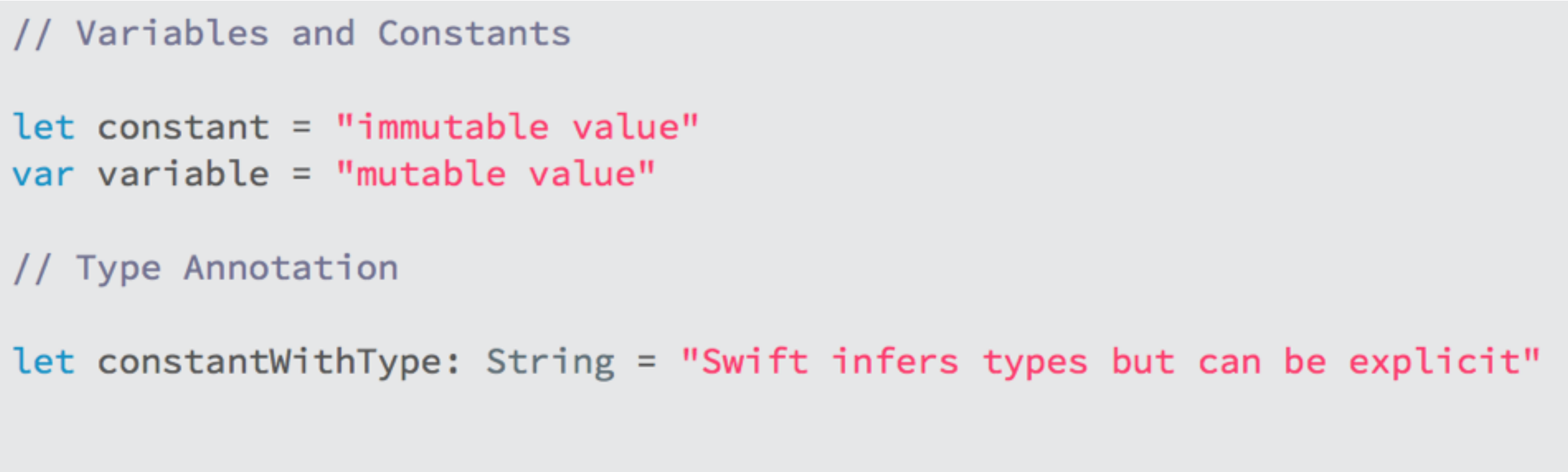
- The main thing that has changed is how you reverse engineer the application

Quick Overview of Swift

What is Swift?

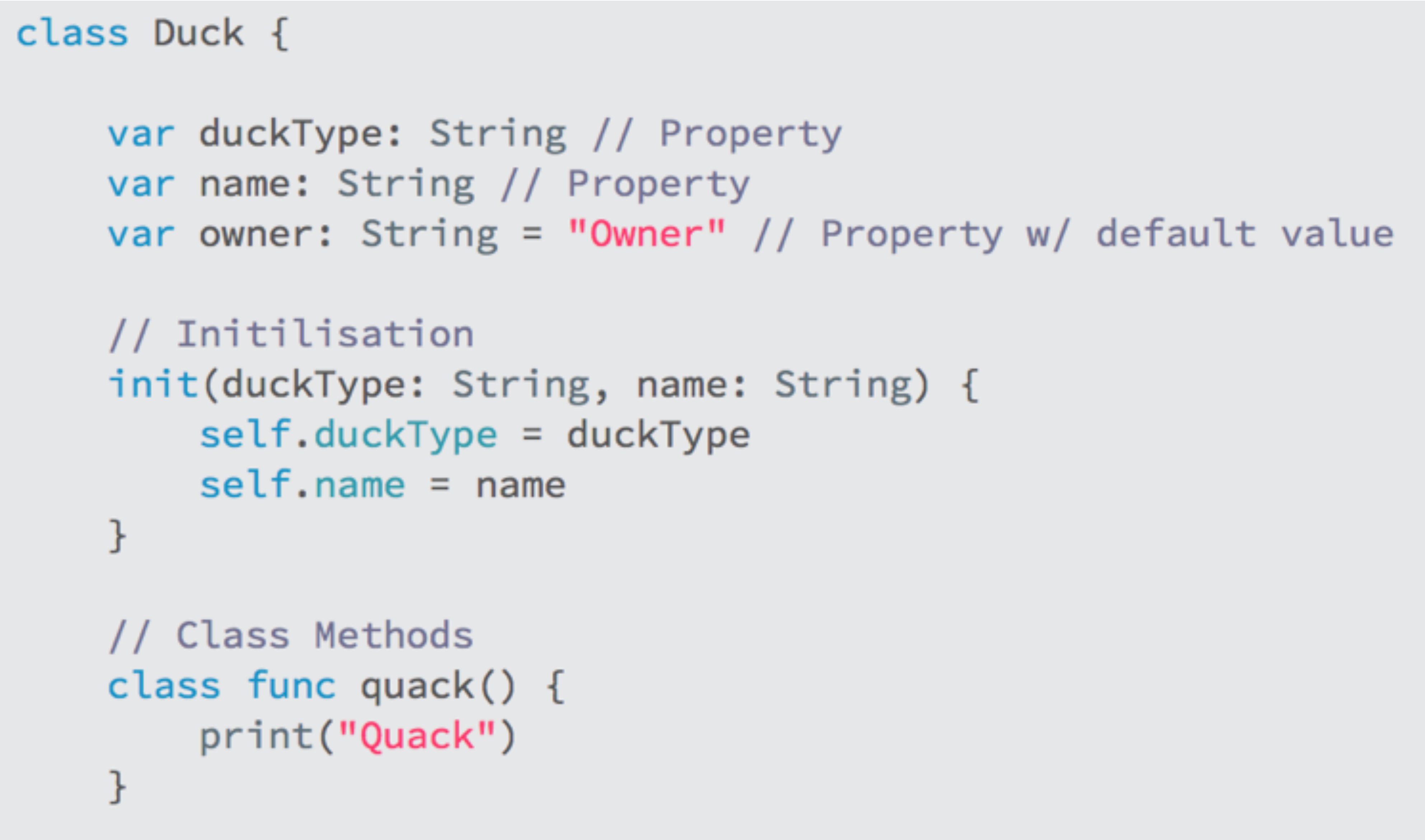
- Compiled language created by Apple
- Released publicly in 2014 at WWDC and has seen multiple revisions since.
- Open source with official implementations for iOS, OS X and Linux.
- Intended to replace Objective-C eventually

Syntax (just the basics to follow along)



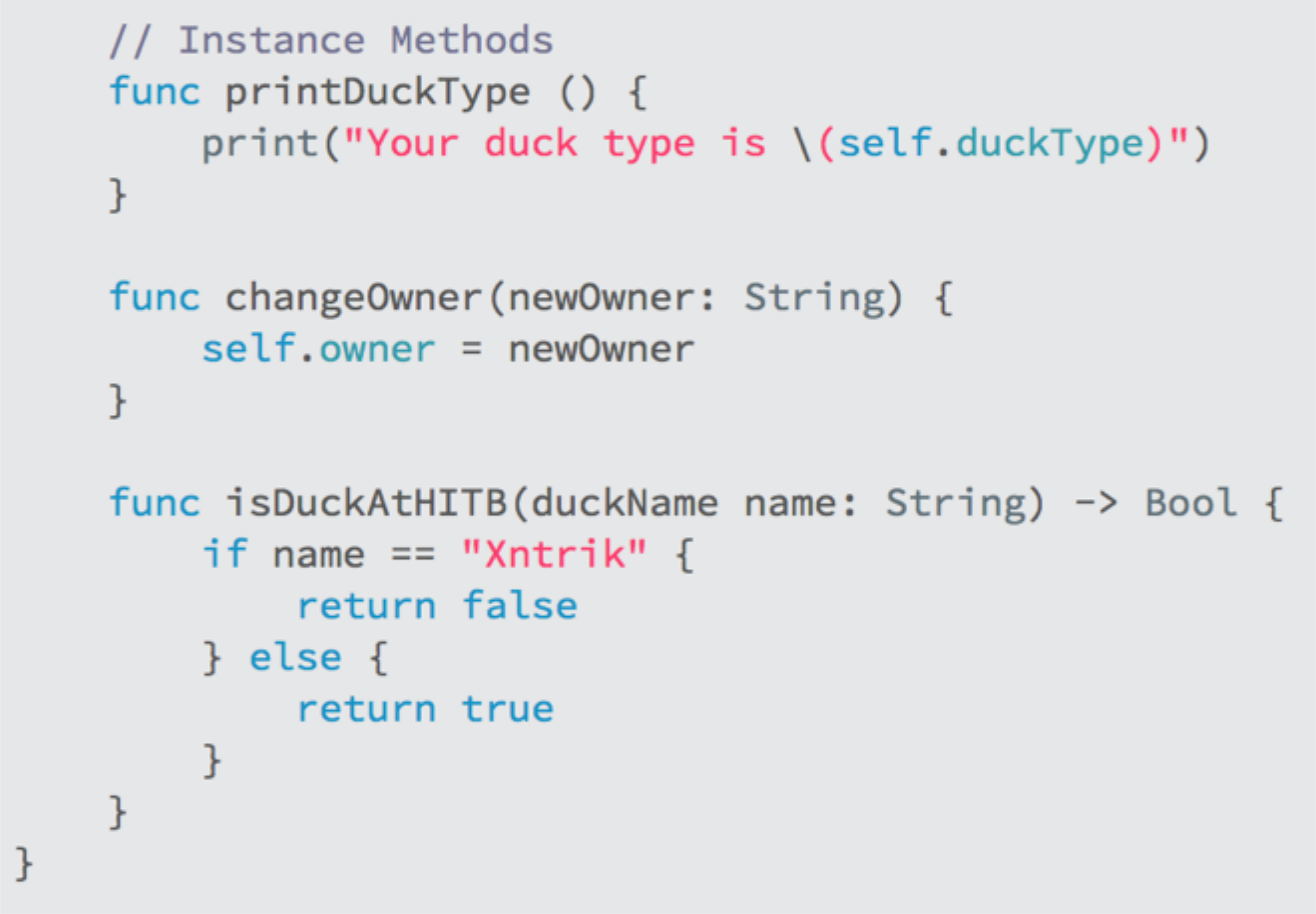
```swift
// Variables and Constants

let constant = "immutable value"
var variable = "mutable value"

// Type Annotation

let constantWithType: String = "Swift infers types but can be explicit"
```

Syntax (just the basics to follow along)



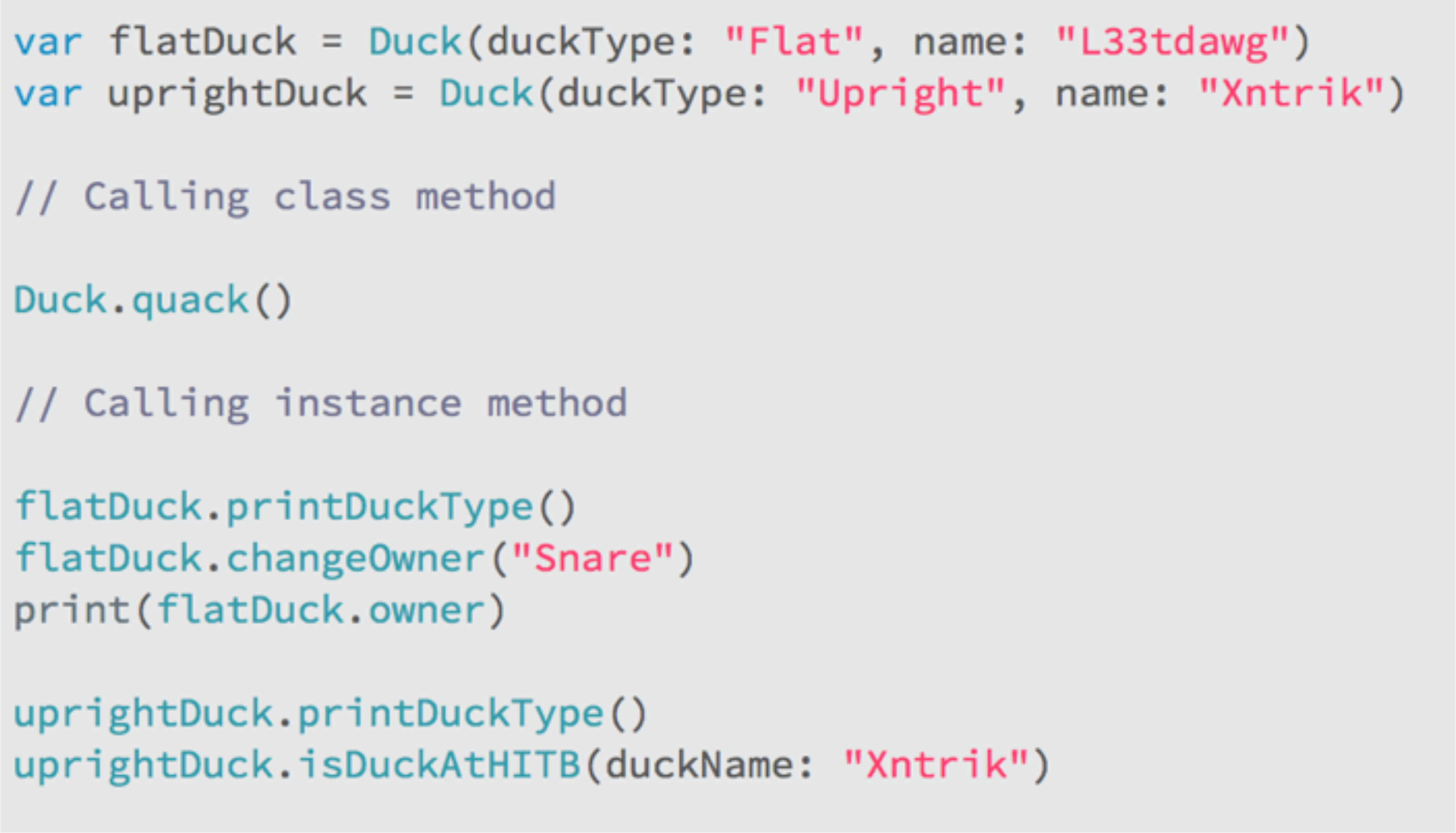
```
class Duck {

    var duckType: String // Property
    var name: String // Property
    var owner: String = "Owner" // Property w/ default value

    // Initilisation
    init(duckType: String, name: String) {
        self.duckType = duckType
        self.name = name
    }

    // Class Methods
    class func quack() {
        print("Quack")
    }
```

Syntax (just the basics to follow along)



```
    // Instance Methods
    func printDuckType () {
        print("Your duck type is \(self.duckType)")
    }

    func changeOwner(newOwner: String) {
        self.owner = newOwner
    }

    func isDuckAtHITB(duckName name: String) -> Bool {
        if name == "Xntrik" {
            return false
        } else {
            return true
        }
    }
}
```

Syntax (just the basics to follow along)



```
var flatDuck = Duck(duckType: "Flat", name: "L33tdawg")
var uprightDuck = Duck(duckType: "Upright", name: "Xntrik")

// Calling class method

Duck.quack()

// Calling instance method

flatDuck.printDuckType()
flatDuck.changeOwner("Snare")
print(flatDuck.owner)

uprightDuck.printDuckType()
uprightDuck.isDuckAtHITB(duckName: "Xntrik")
```

Types

- All basic C and Objective-C types -> String, Bool, Int , Float etc.

- Collection Types -> Array, Set, Dictionary

- Optional Types -> works with all types, no more nil pointers like Objective-C

- Swift is a type safe language

Objective-C Compatibility

- Objective-C compatibility and interoperability
    - Uses the same runtime environment
    - Still supports C and C++ in the same app but can't be called from Swift like Objective-C
    - Can allow for some dynamic features and runtime manipulation

Other Language Features

- Barely scratched the surface
  - Structs, Protocols, Extensions, Closures, Enumerations, Optionals, Generics, Type Casting, Access Control, Error Handling, Assertions….
  - Automatic Reference Counting
  - Unicode…
    - var 💩 = 💩💩💩💩💩()

Other Language Features

The Swift Programming Language

Swift 2.2 Edition

Challenges Reversing Swift Apps

Challenges

- Less dynamic than Objective-C
    - Less flexible than Objective-C in some areas
    - Can make it harder to do some of the standard tasks you would do on a standard app pen test
    - Less of an issue now because most Swift apps will include be mixed with Objective-C
- Limited tooling
    - We will explore this in more detail

Challenges

- Rapidly evolving syntax, APIs and features and Apple doesn't care too much about breaking changes.
  - v1.0 - September 2014
  - v1.1 - October 2014
  - v1.2 - April 2015
  - v2.0 - September 2015 (Open Sourced, Linux)
  - v2.2 - March 2016
  - v3.0 - Late 2016

Reversing Swift Apps

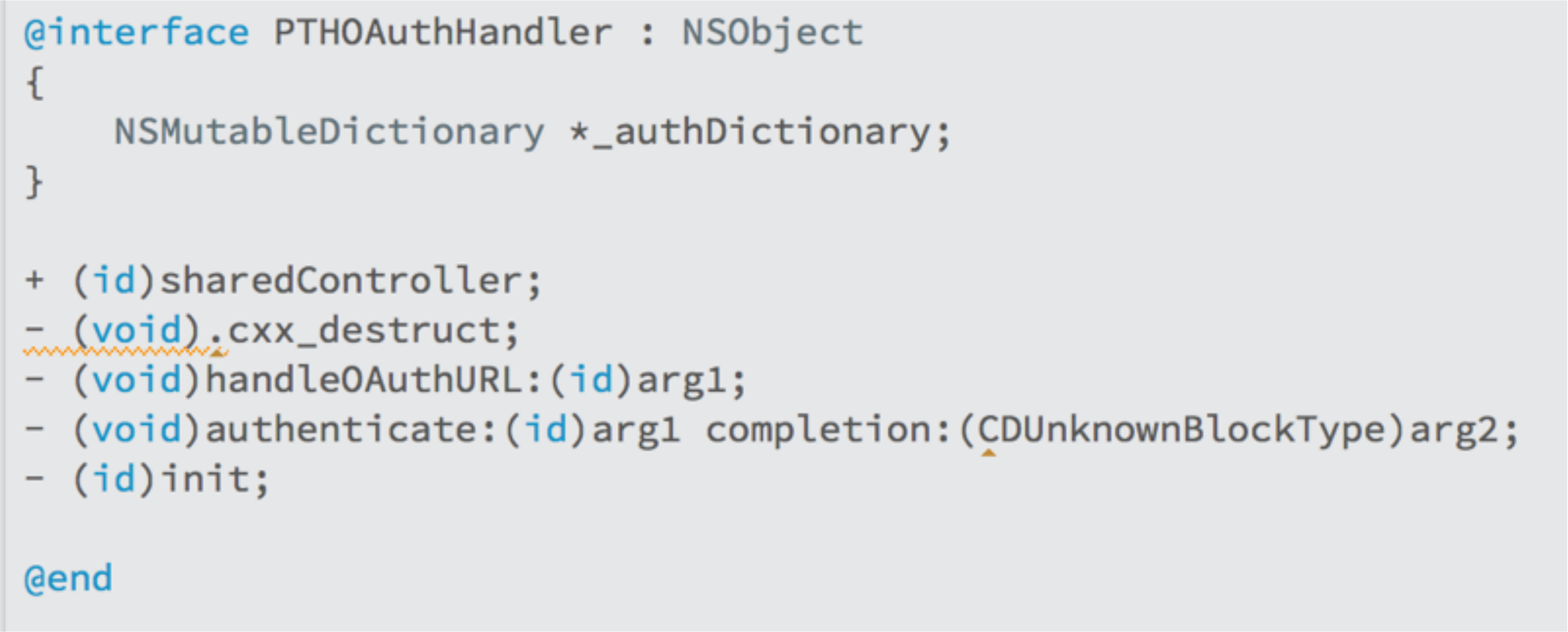
- Two primary reverse engineering activities when conducting a “black box” pen test
  - Dumping and analysing class information from the binary
  - Retrieving information at runtime using debuggers, function hooking, tracing etc.

Retrieving Class Information

Class Dump?

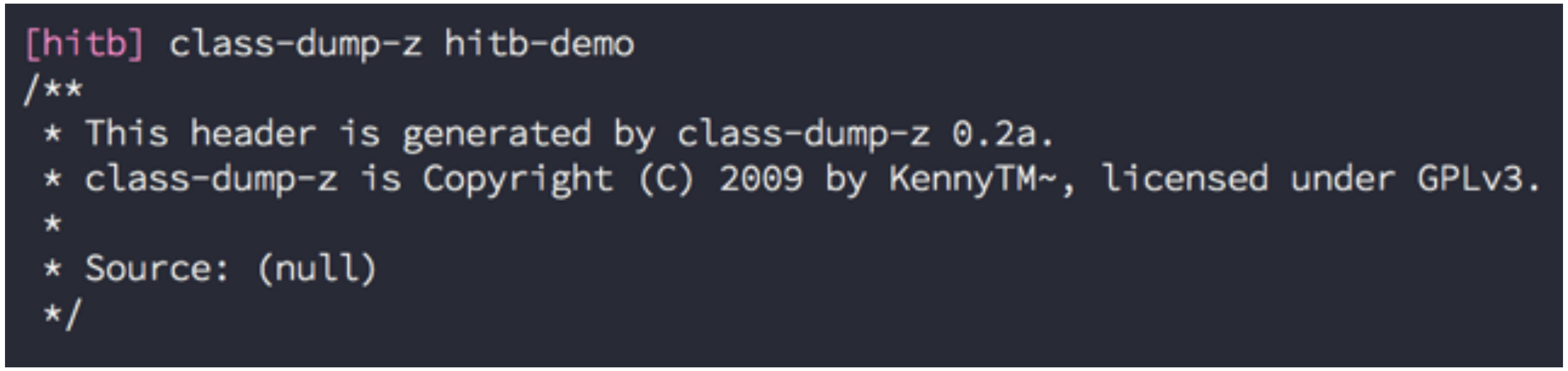
- The most common and easiest way to retrieve class data from an Objective-C binary is the class-dump utility
- class-dump retrieves class information and formats to look like the equivalent of an Objective-C header file
- Usually one of the first things you do when looking at an app

Class Dump?

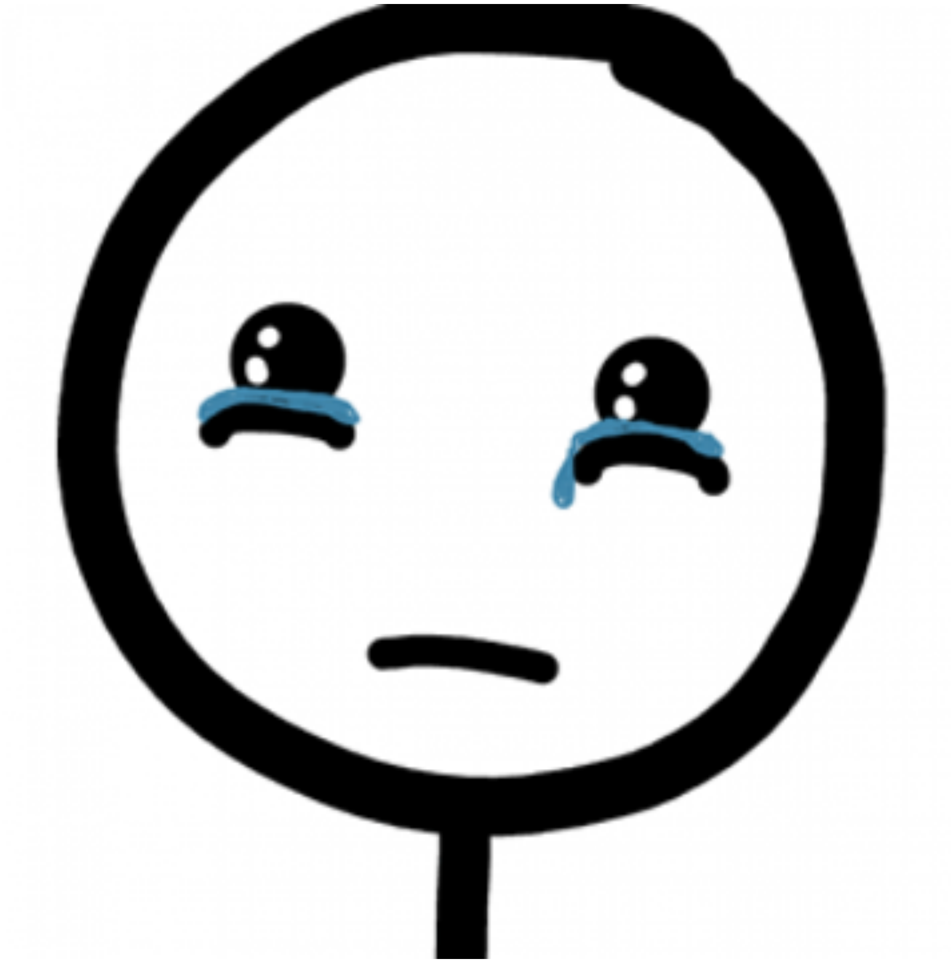


```
@interface PTHOAuthHandler : NSObject
{
    NSMutableDictionary *_authDictionary;
}

+ (id)sharedController;
- (void).cxx_destruct;
- (void)handleOAuthURL:(id)arg1;
- (void)authenticate:(id)arg1 completion:(CDUnknownBlockType)arg2;
- (id)init;

@end
```

Class Dump?



```
[hitb] class-dump-z hitb-demo
/**
 * This header is generated by class-dump-z 0.2a.
 * class-dump-z is Copyright (C) 2009 by KennyTM~, licensed under GPLv3.
 *
 * Source: (null)
 */
```

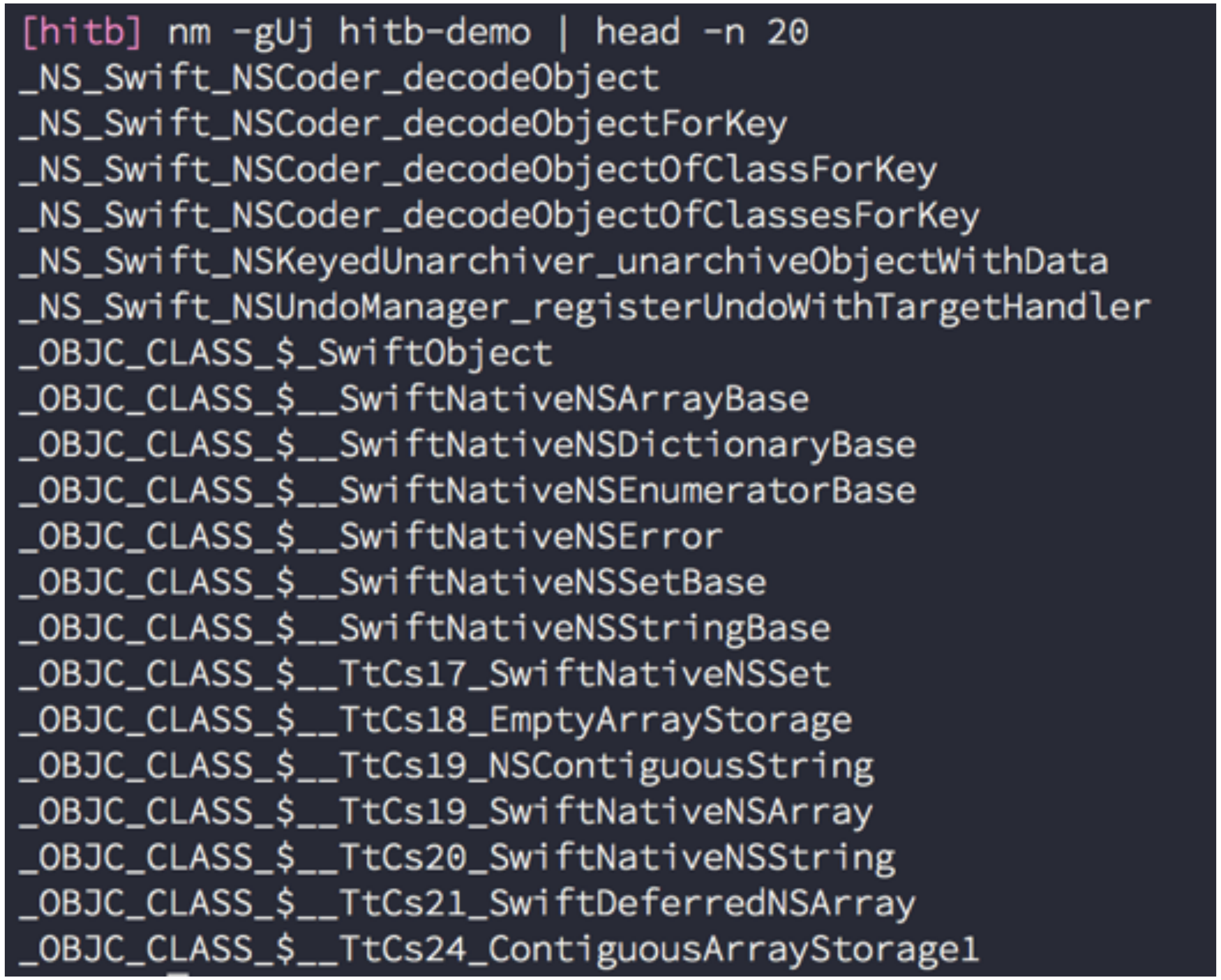


What next?

- So class-dump-z doesn’t work with Swift binaries :(
- Now what?
- Let’s start diving into the binary

Symbol Table

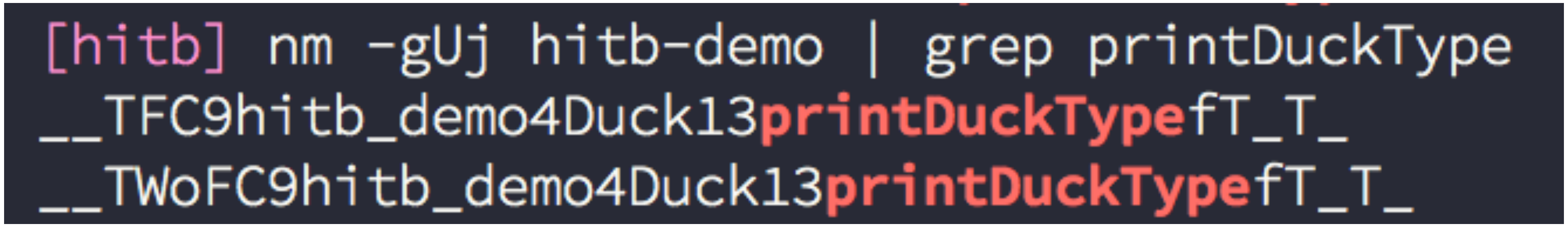
- What do we get if we dump the symbol table?



```
[hitb] nm -gUj hitb-demo | head -n 20
_NS_Swift_NSCoder_decodeObject
_NS_Swift_NSCoder_decodeObjectForKey
_NS_Swift_NSCoder_decodeObjectOfClassForKey
_NS_Swift_NSCoder_decodeObjectOfClassesForKey
_NS_Swift_NSKeyedUnarchiver_unarchiveObjectWithData
_NS_Swift_NSUndoManager_registerUndoWithTargetHandler
_OBJC_CLASS_$_SwiftObject
_OBJC_CLASS_$__SwiftNativeNSArrayBase
_OBJC_CLASS_$__SwiftNativeNSDictionaryBase
_OBJC_CLASS_$__SwiftNativeNSEnumeratorBase
_OBJC_CLASS_$__SwiftNativeNSError
_OBJC_CLASS_$__SwiftNativeNSSetBase
_OBJC_CLASS_$__SwiftNativeNSStringBase
_OBJC_CLASS_$__TtCs17_SwiftNativeNSSet
_OBJC_CLASS_$__TtCs18_EmptyArrayStorage
_OBJC_CLASS_$__TtCs19_NSContiguousString
_OBJC_CLASS_$__TtCs19_SwiftNativeNSArray
_OBJC_CLASS_$__TtCs20_SwiftNativeNSString
_OBJC_CLASS_$__TtCs21_SwiftDeferredNSArray
_OBJC_CLASS_$__TtCs24_ContiguousArrayStorage1
```

Symbol Table

- What if we look for something we know is in the binary?



```
[hitb] nm -gUj hitb-demo | grep printDuckType
__TFC9hitb_demo4Duck13printDuckTypefT_T_
__TWoFC9hitb_demo4Duck13printDuckTypefT_T_
```

Name Mangling

- Looks promising but it’s a far cry from the output of class-dump and is kind of hard to make out

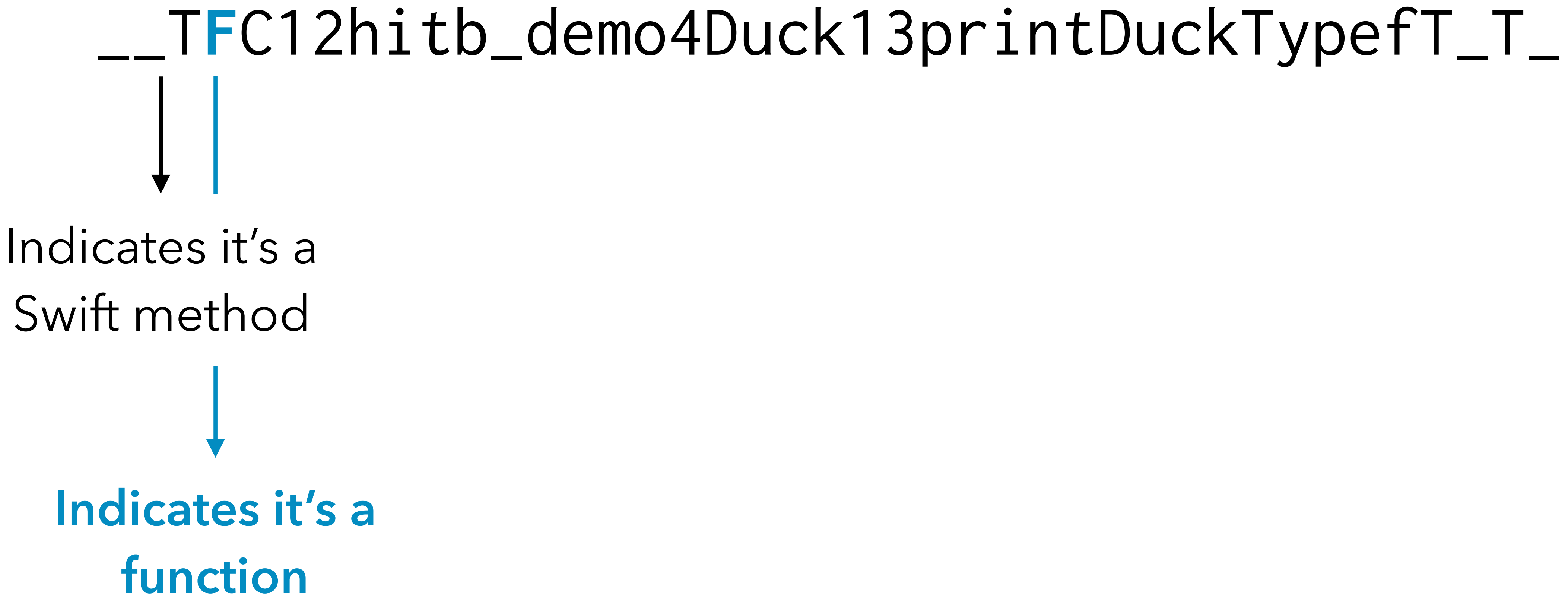
- Swift stores metadata about a function in it’s symbols in the process “mangling” the name.

Name Mangling

```
__TFC12hitb_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Name Mangling

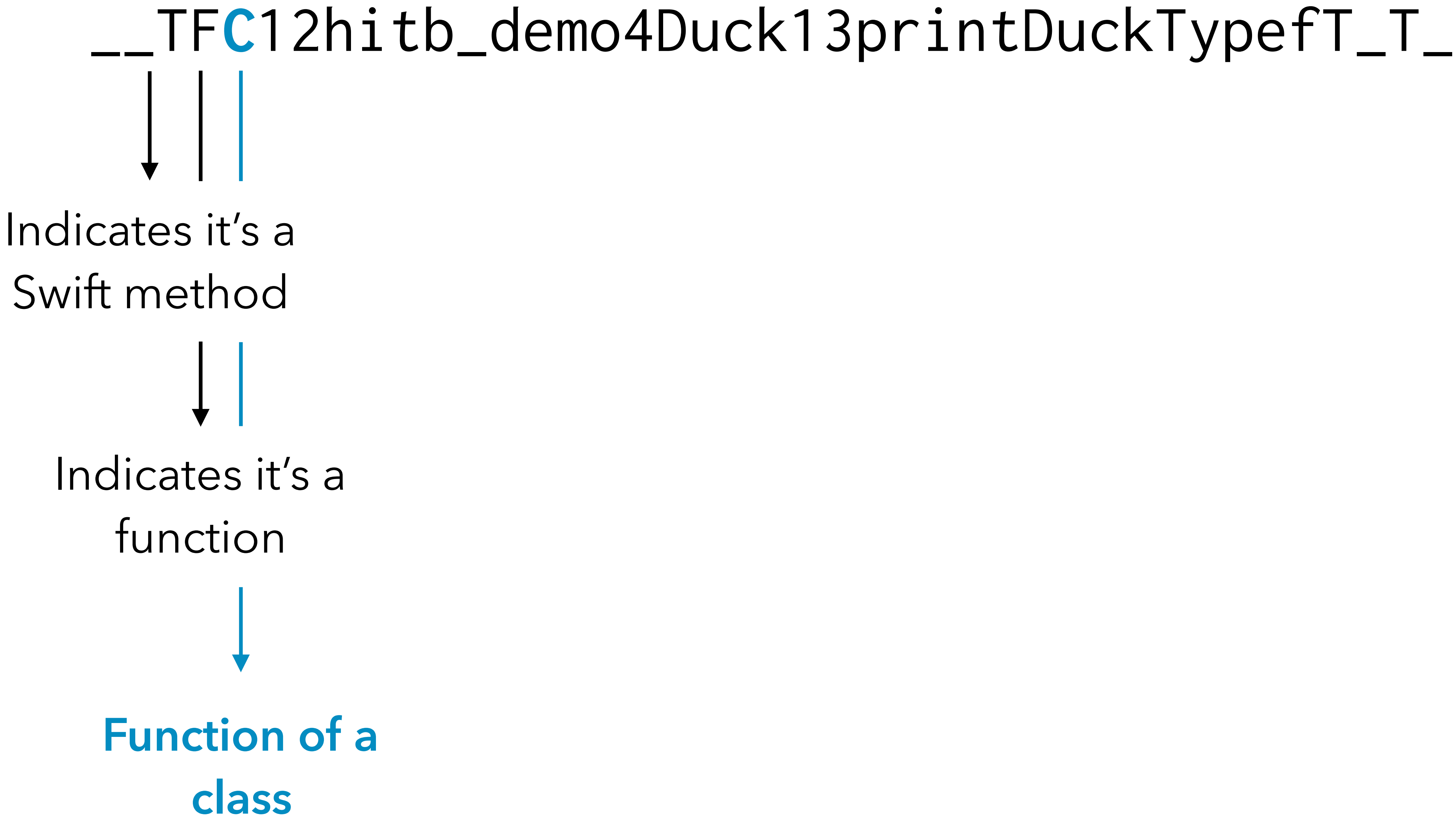


`__TFC12hitb_demo4Duck13printDuckTypefT_T_`

Indicates it's a Swift method

Indicates it's a function

Name Mangling



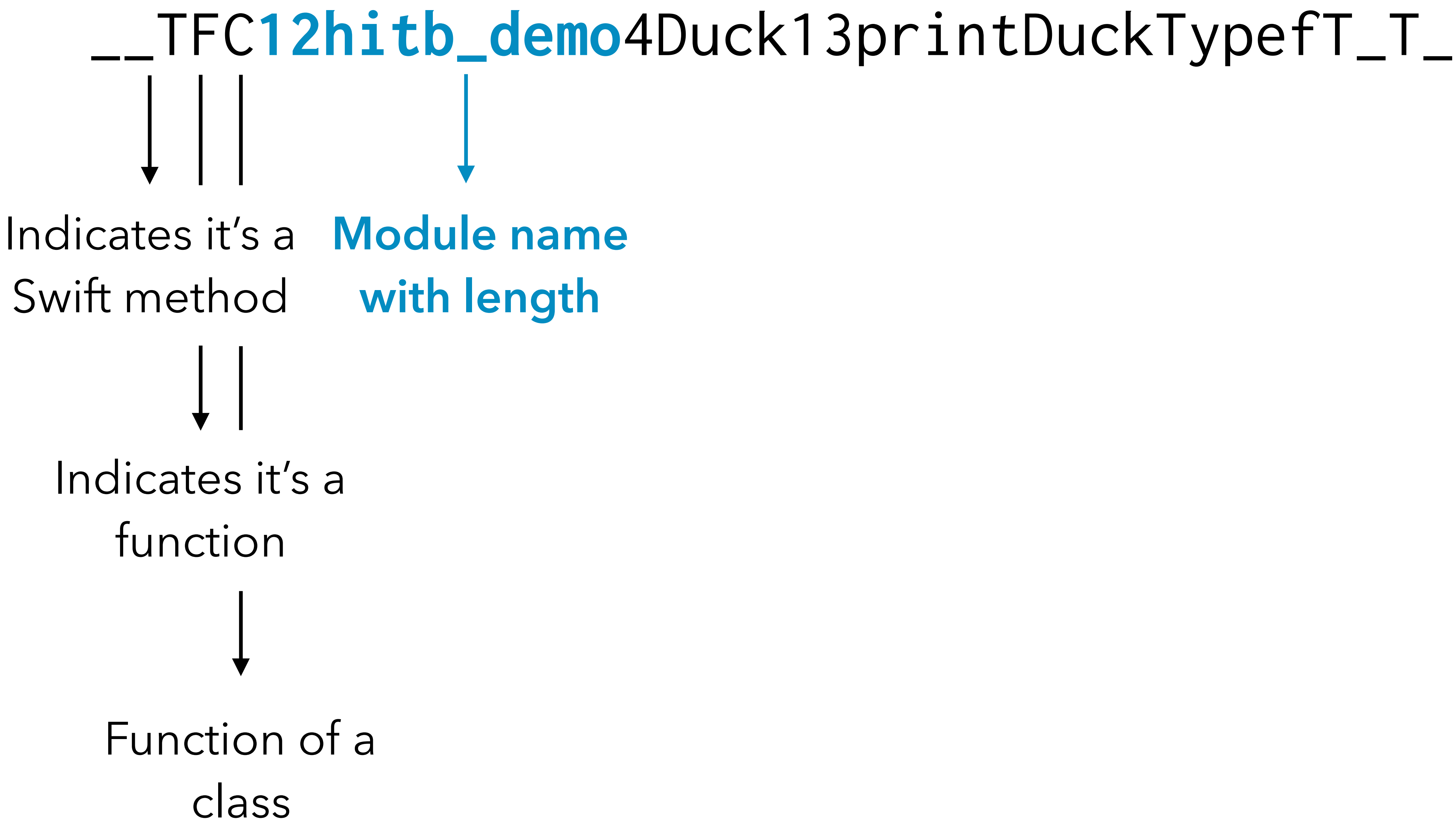
```
__TFC12hitb_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Indicates it's a function

Function of a class

Name Mangling



```
__TFC12hitb_demo4Duck13printDuckTypefT_T_
```

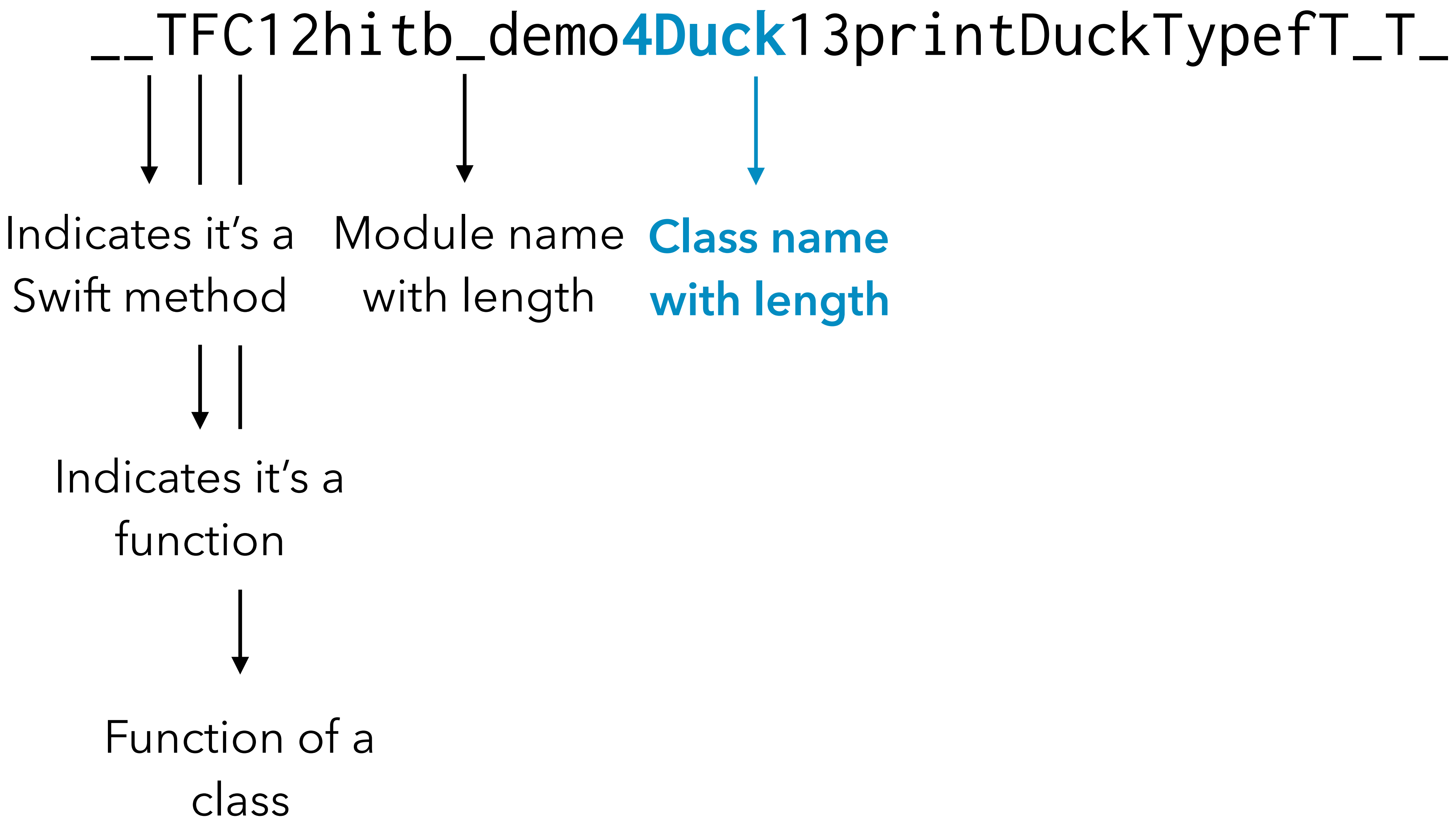
Indicates it's a Swift method

Module name with length

Indicates it's a function

Function of a class

Name Mangling



__TFC12hitb_demo**4Duck**13printDuckTypefT_T_

Indicates it's a Swift method

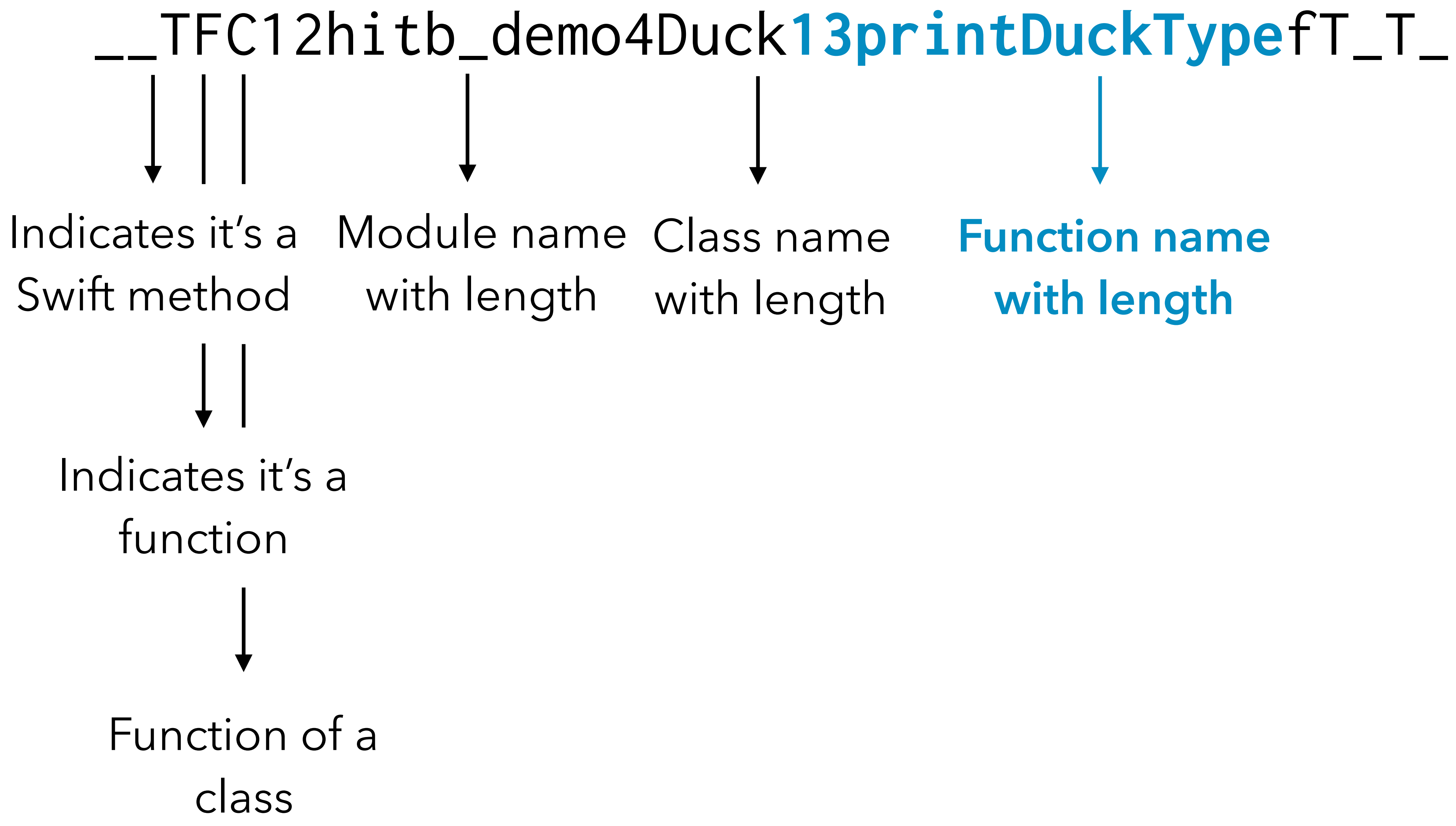
Module name with length

Class name with length

Indicates it's a function

Function of a class

Name Mangling



```
__TFC12hitb_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Module name with length

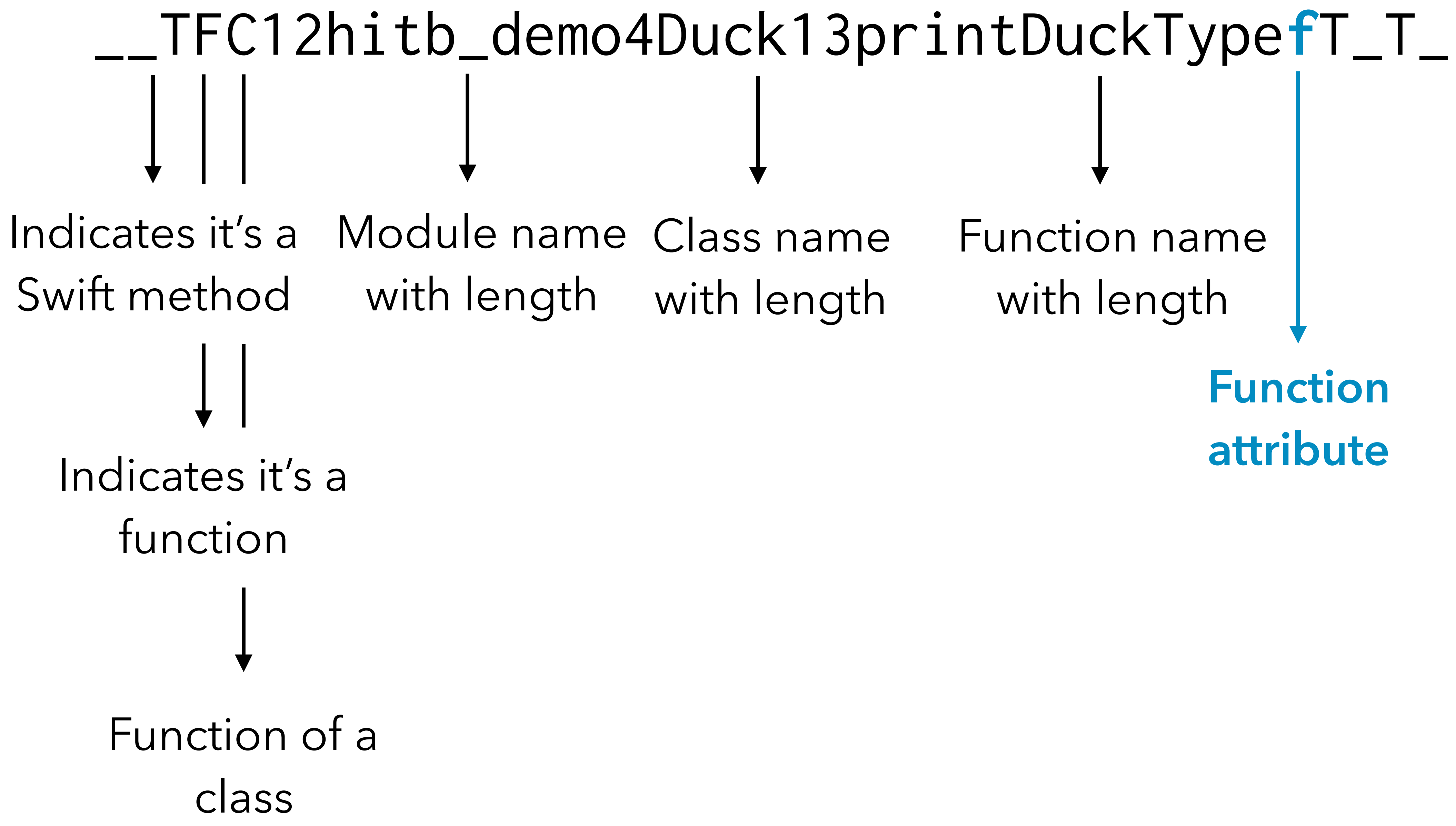
Class name with length

Function name with length

Indicates it's a function

Function of a class

Name Mangling



```
__TFC12hitb_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Module name with length

Class name with length

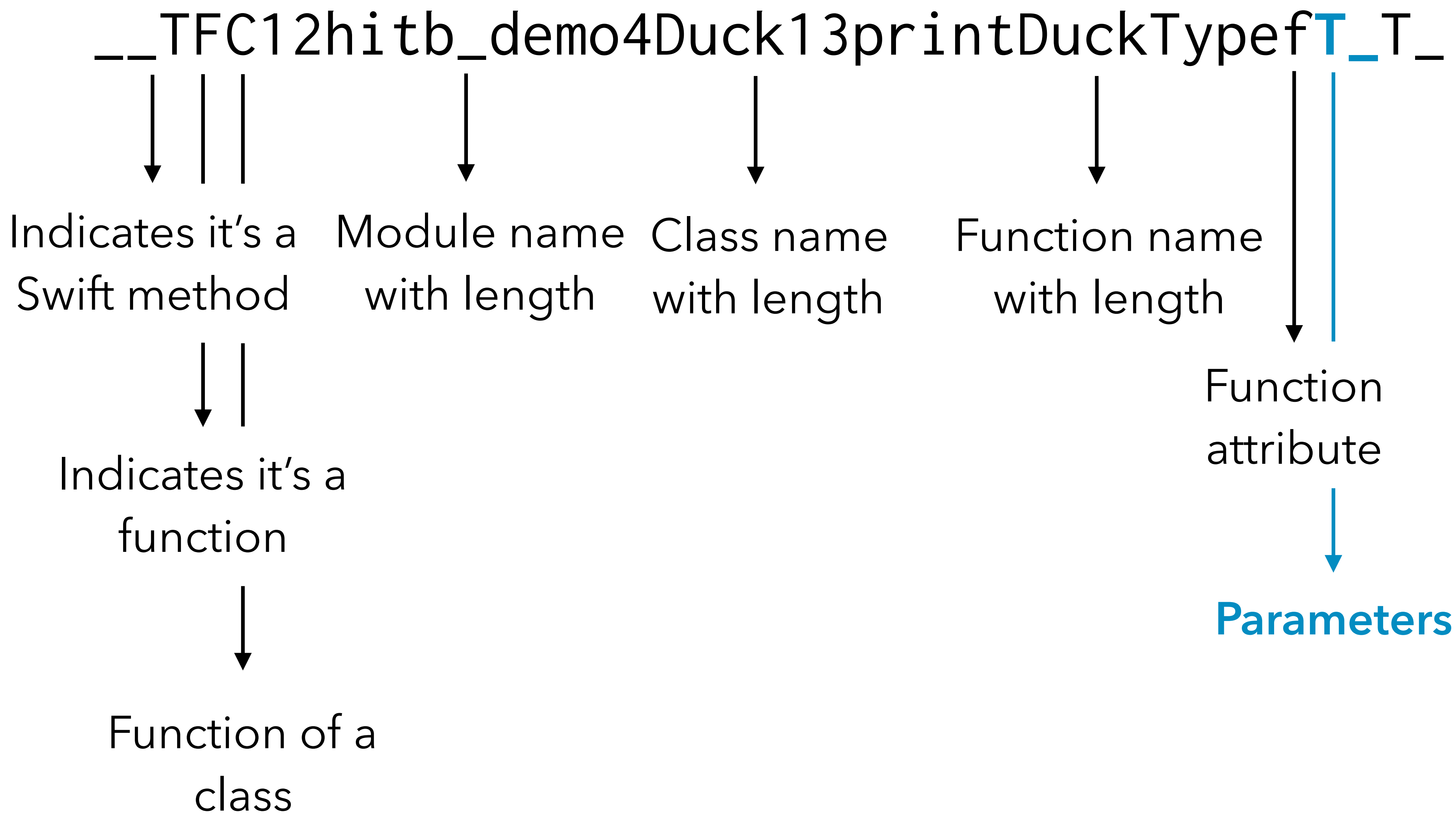
Function name with length

Function attribute

Indicates it's a function

Function of a class

Name Mangling



```
__TFC12hitb_demo4Duck13printDuckTypefT_T_
```

Indicates it's a Swift method

Module name with length

Class name with length

Function name with length

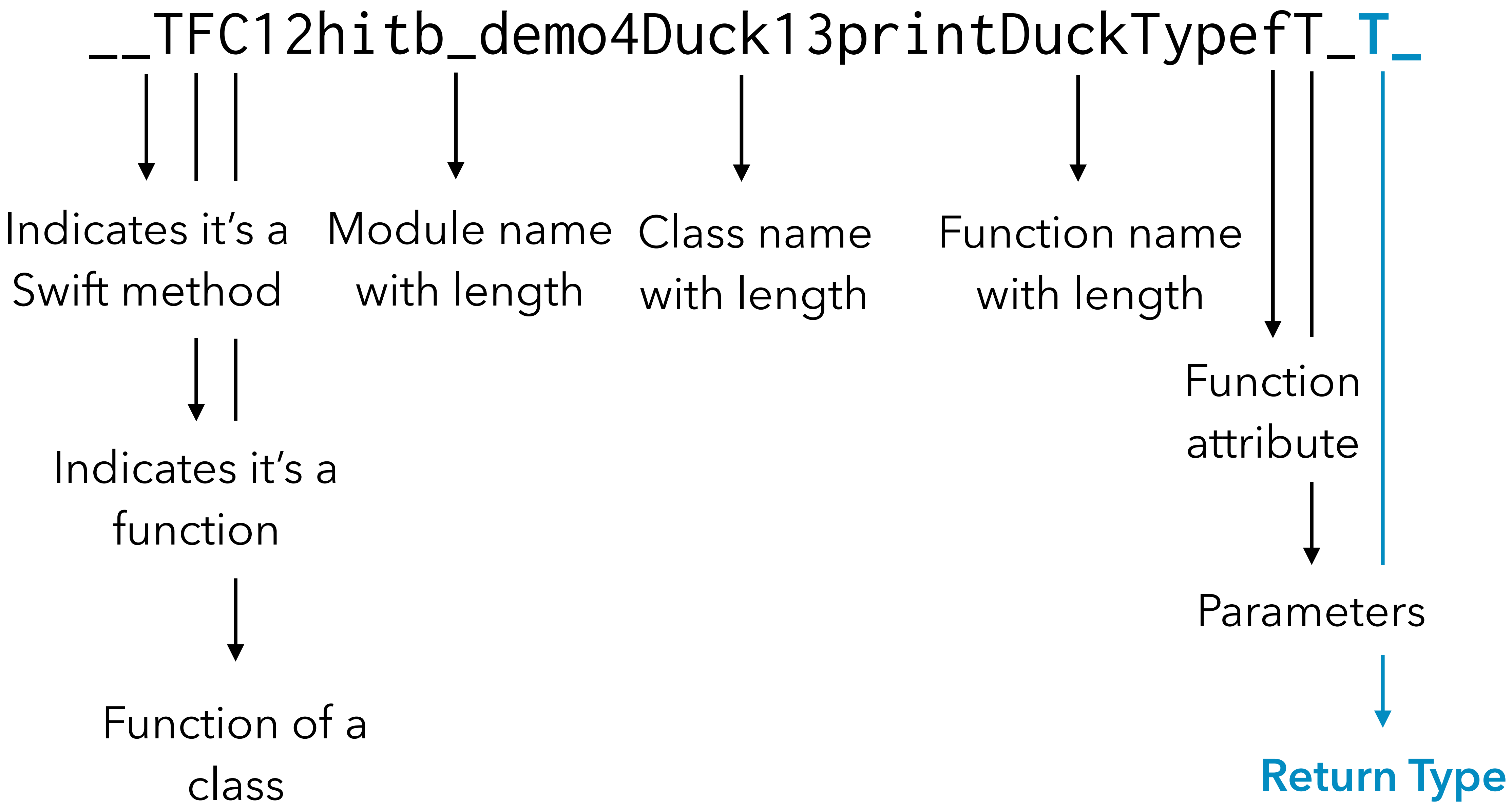
Indicates it's a function

Function attribute

Parameters

Function of a class

Name Mangling



`__TFC12hitb_demo4Duck13printDuckTypefT_T_`

Indicates it's a Swift method

Module name with length

Class name with length

Function name with length

Indicates it's a function

Function attribute

Function of a class

Parameters

Return Type

Trustwave® SpiderLabs®

Function Attributes

f	Normal function
s	Setter
g	Getter
d	Destructor
D	Deallocator
c	Constructor
C	Allocator

Return Types

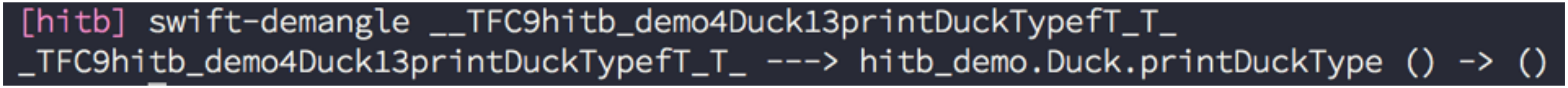
a	Array
b	Boolean
c	Unicode Scalar
d	Double
f	Float
i	Integer
u	Unsigned Integer
Q	Implicitly Unwrapped Optional
S	String

swift-demangle

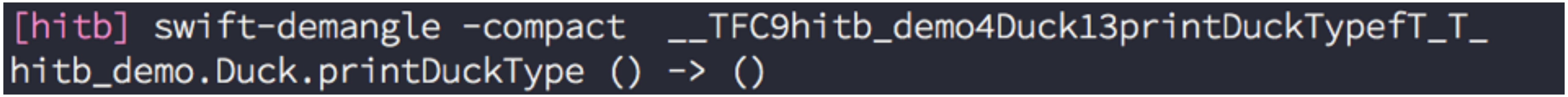
- So now we know roughly the way the names are mangle you could use this to create a script that "de-mangles" the names

- Apple has already thought of that and includes a utility called swift-demangle to do just that

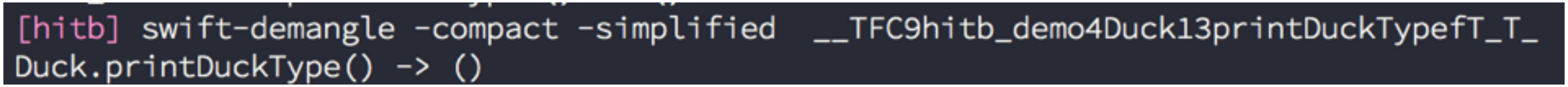
swift-demangle



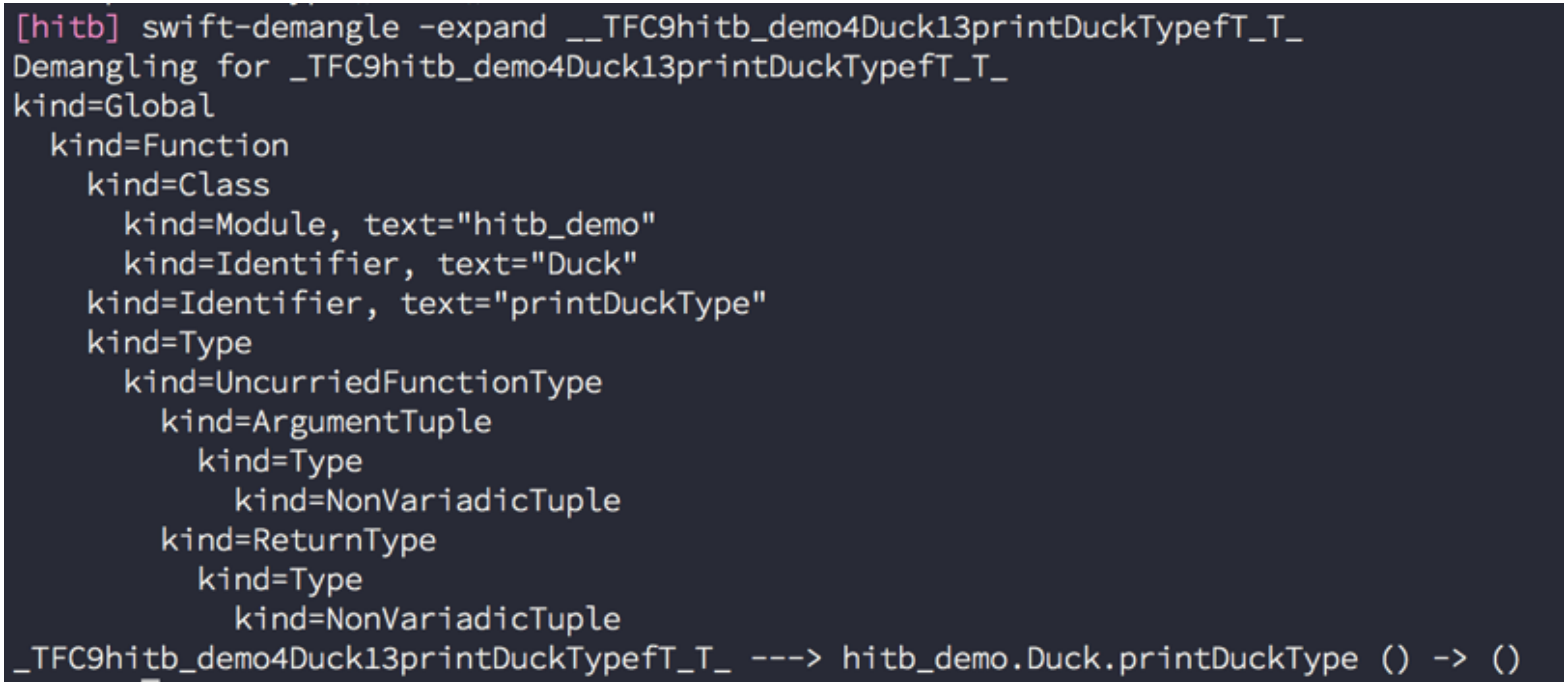
```
[hitb] swift-demangle __TFC9hitb_demo4Duck13printDuckTypefT_T_
_TFC9hitb_demo4Duck13printDuckTypefT_T_ ---> hitb_demo.Duck.printDuckType () -> ()
```



```
[hitb] swift-demangle -compact  __TFC9hitb_demo4Duck13printDuckTypefT_T_
hitb_demo.Duck.printDuckType () -> ()
```



```
[hitb] swift-demangle -compact -simplified  __TFC9hitb_demo4Duck13printDuckTypefT_T_
Duck.printDuckType() -> ()
```



```
[hitb] swift-demangle -expand __TFC9hitb_demo4Duck13printDuckTypefT_T_
Demangling for _TFC9hitb_demo4Duck13printDuckTypefT_T_
kind=Global
  kind=Function
    kind=Class
      kind=Module, text="hitb_demo"
      kind=Identifier, text="Duck"
    kind=Identifier, text="printDuckType"
    kind=Type
      kind=UncurriedFunctionType
        kind=ArgumentTuple
          kind=Type
            kind=NonVariadicTuple
        kind=ReturnType
          kind=Type
            kind=NonVariadicTuple
_TFC9hitb_demo4Duck13printDuckTypefT_T_ ---> hitb_demo.Duck.printDuckType () -> ()
```

swift-demangle

- With nm and swift-demangle and some shell scripting you should be able to easily grab the function signatures from an app

- Should be all you need to get basically the same information you would from class-dump to start assessing the app

class-dump-s

- Hacked together script that demangles names and formats the output to approximate the output of class-dump
- Written in Swift
- Demo

Stripped Binaries

- CAVEAT: If the developer stripped symbols from the binary then these techniques obviously won’t work.

- Reverse engineering stripped binaries is a bit more complicated

Objective-C Compatibility

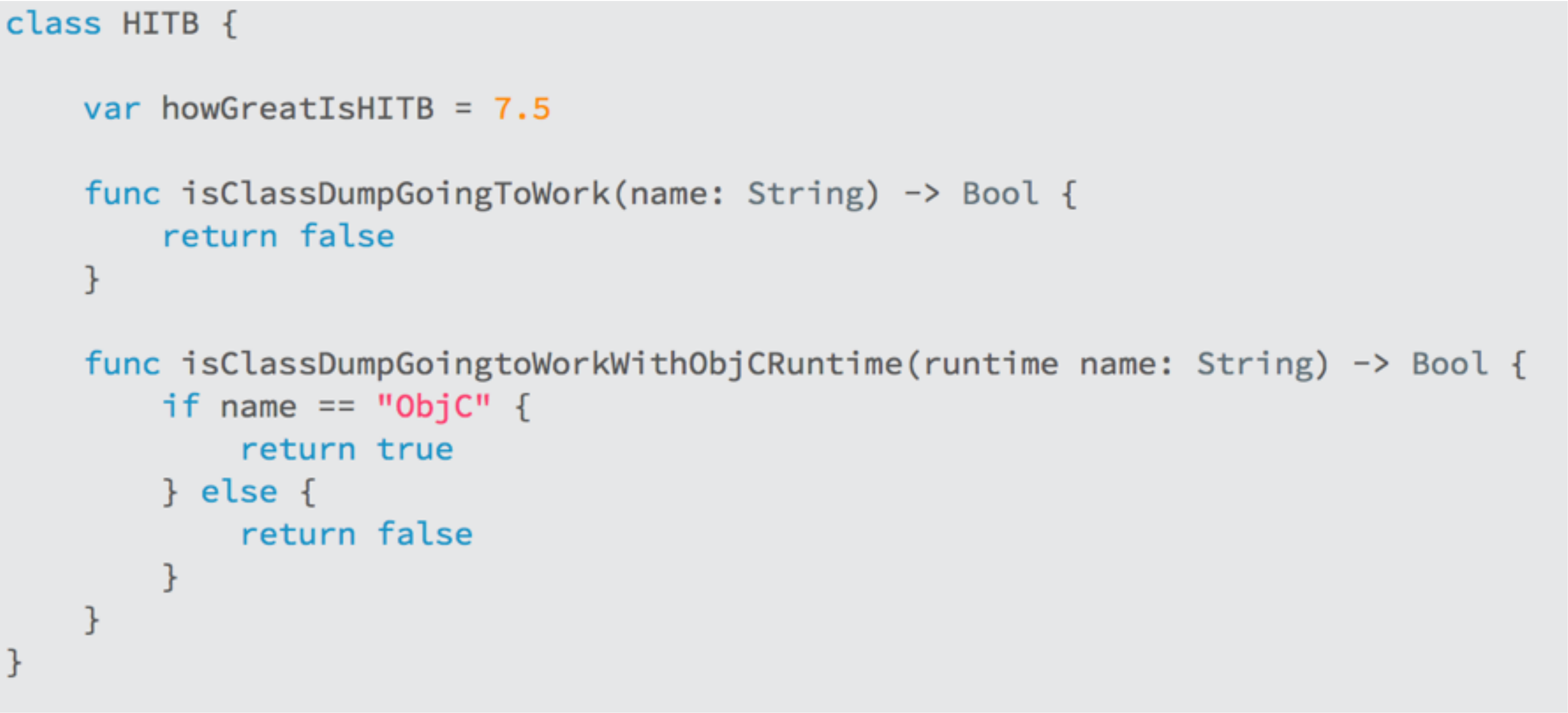
- Part of the reason it’s much easier to get class information from Objective-C binaries is because it’s necessary for the Objective-C runtime to have that info

- So what happens when you import Objective-C frameworks or use Objective-C in your app?

Revisiting Class Dump

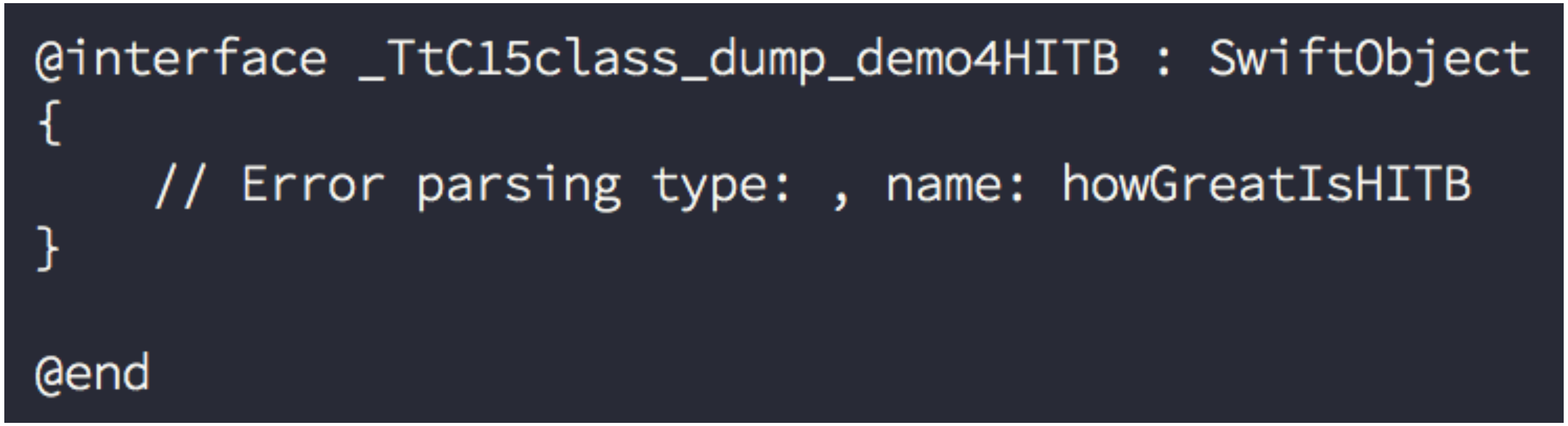
- The latest branch of class-dump by Steven Nygard (the original class-dump utility) has limited support for Swift.
- Need to download and build from source (no binary release yet)
- https://github.com/nygard/class-dump

Revisiting Class Dump

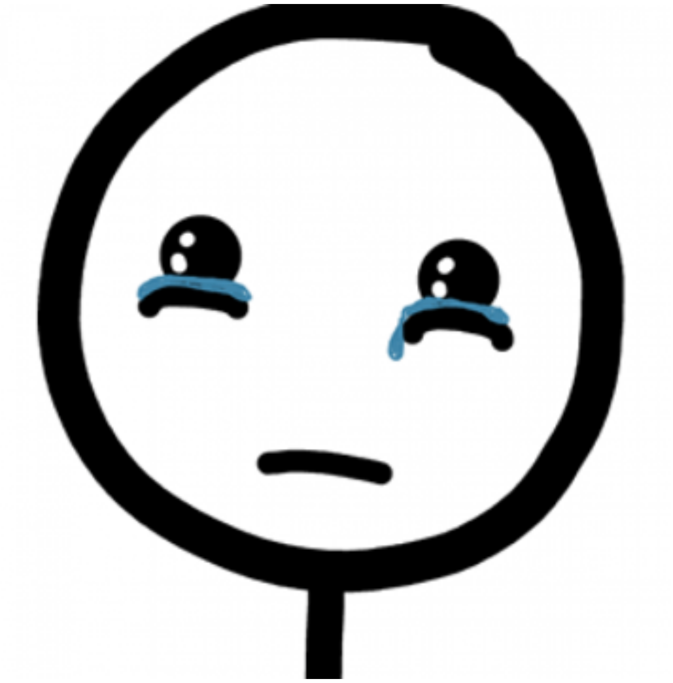


```
class HITB {

    var howGreatIsHITB = 7.5

    func isClassDumpGoingToWork(name: String) -> Bool {
        return false
    }

    func isClassDumpGoingtoWorkWithObjCRuntime(runtime name: String) -> Bool {
        if name == "ObjC" {
            return true
        } else {
            return false
        }
    }
}
```

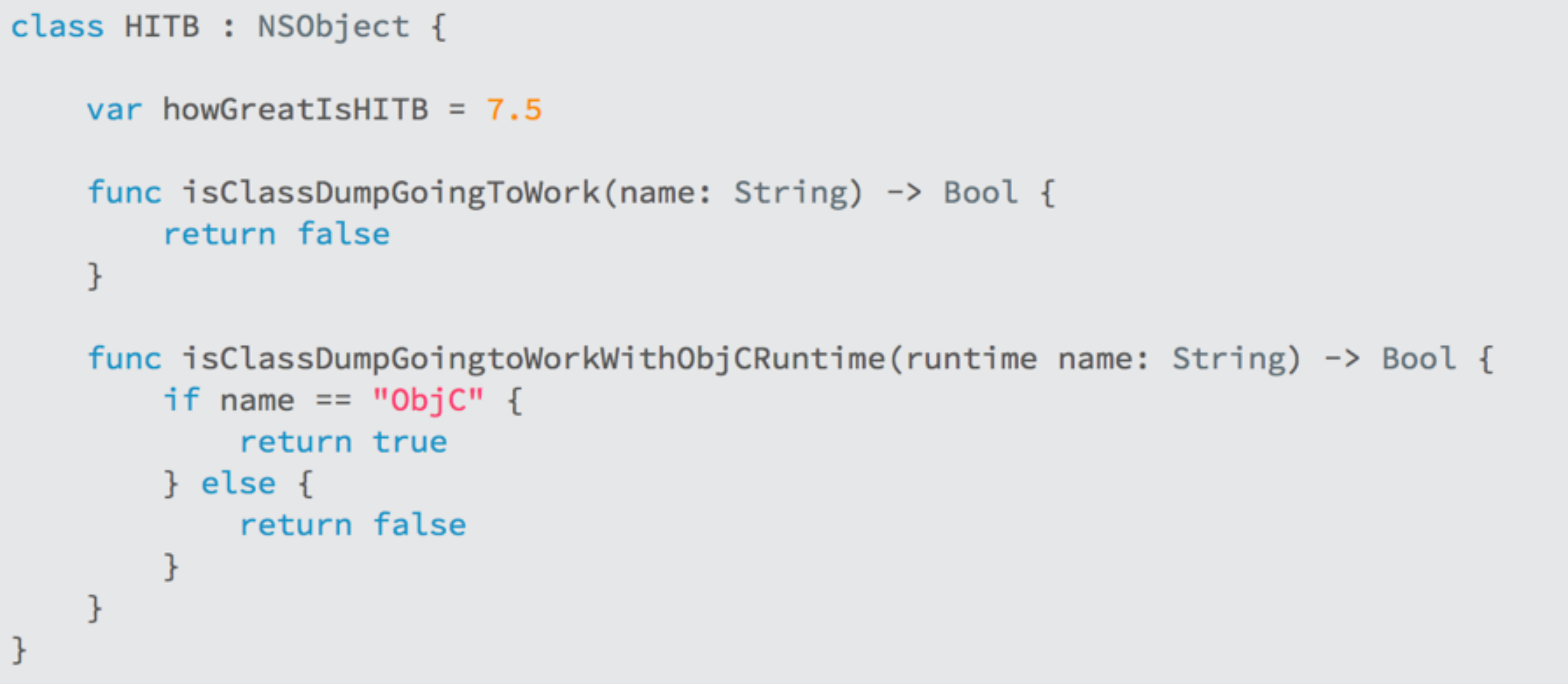
Revisiting Class Dump



```
@interface _TtC15class_dump_demo4HITB : SwiftObject
{
    // Error parsing type: , name: howGreatIsHITB
}

@end
```

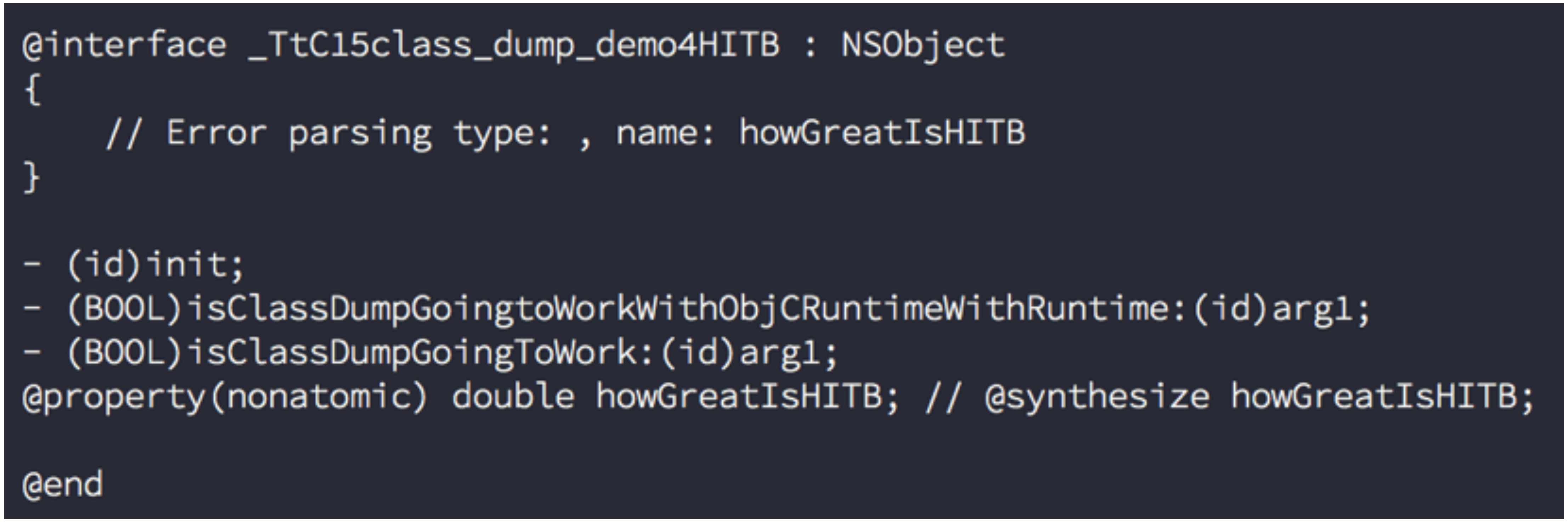


Revisiting Class Dump

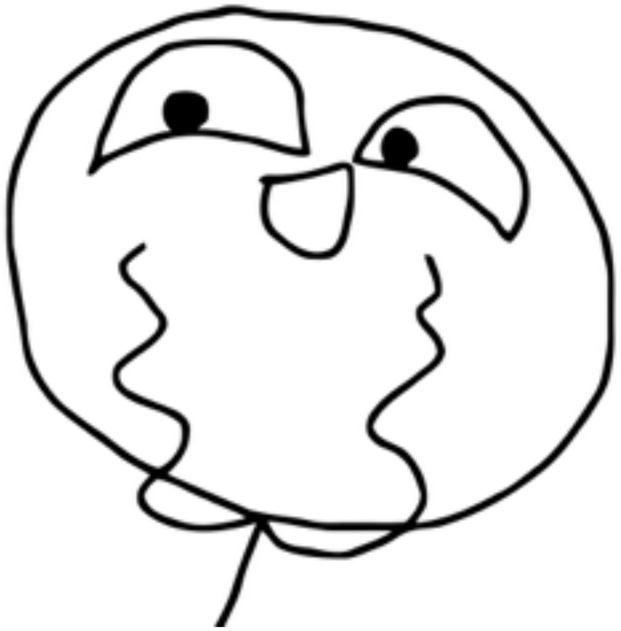


```
class HITB : NSObject {

    var howGreatIsHITB = 7.5

    func isClassDumpGoingToWork(name: String) -> Bool {
        return false
    }

    func isClassDumpGoingtoWorkWithObjCRuntime(runtime name: String) -> Bool {
        if name == "ObjC" {
            return true
        } else {
            return false
        }
    }
}
```

Revisiting Class Dump



```
@interface _TtC15class_dump_demo4HITB : NSObject
{
    // Error parsing type: , name: howGreatIsHITB
}

- (id)init;
- (BOOL)isClassDumpGoingtoWorkWithObjCRuntimeWithRuntime:(id)arg1;
- (BOOL)isClassDumpGoingToWork:(id)arg1;
@property(nonatomic) double howGreatIsHITB; // @synthesize howGreatIsHITB;

@end
```



Other Options

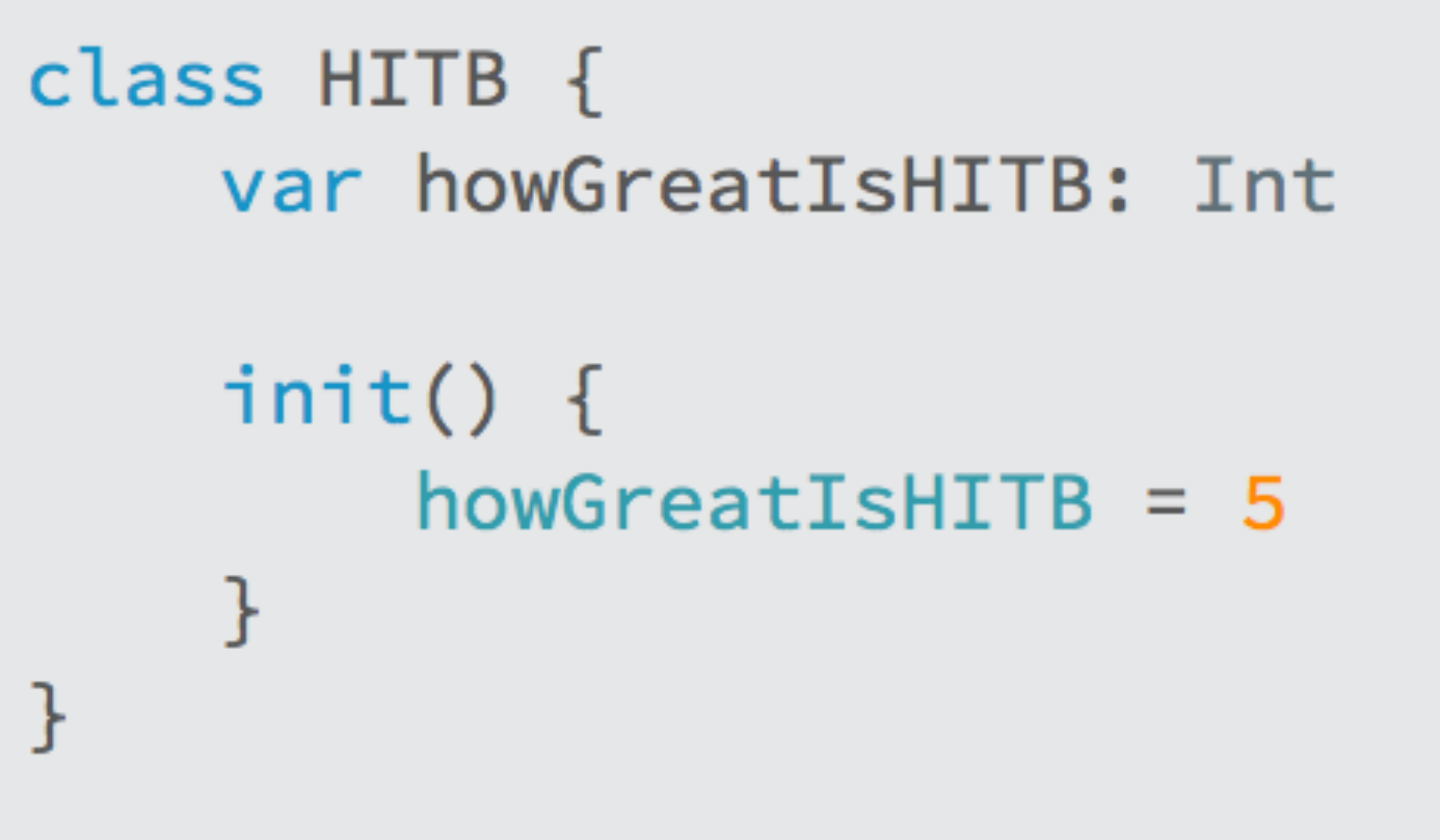
- Disassemblers (i.e. Hopper, IDA Pro)
  - Necessary for lower level insight into the app
  - To demangle Swift function names https://github.com/Januzellij/hopperscripts

Function Hooking

Hooking Swift Methods

- Still possible.
- Much easier with in mixed Swift/Objective-C binaries.
- Can still write tweaks with Mobile Substrate.

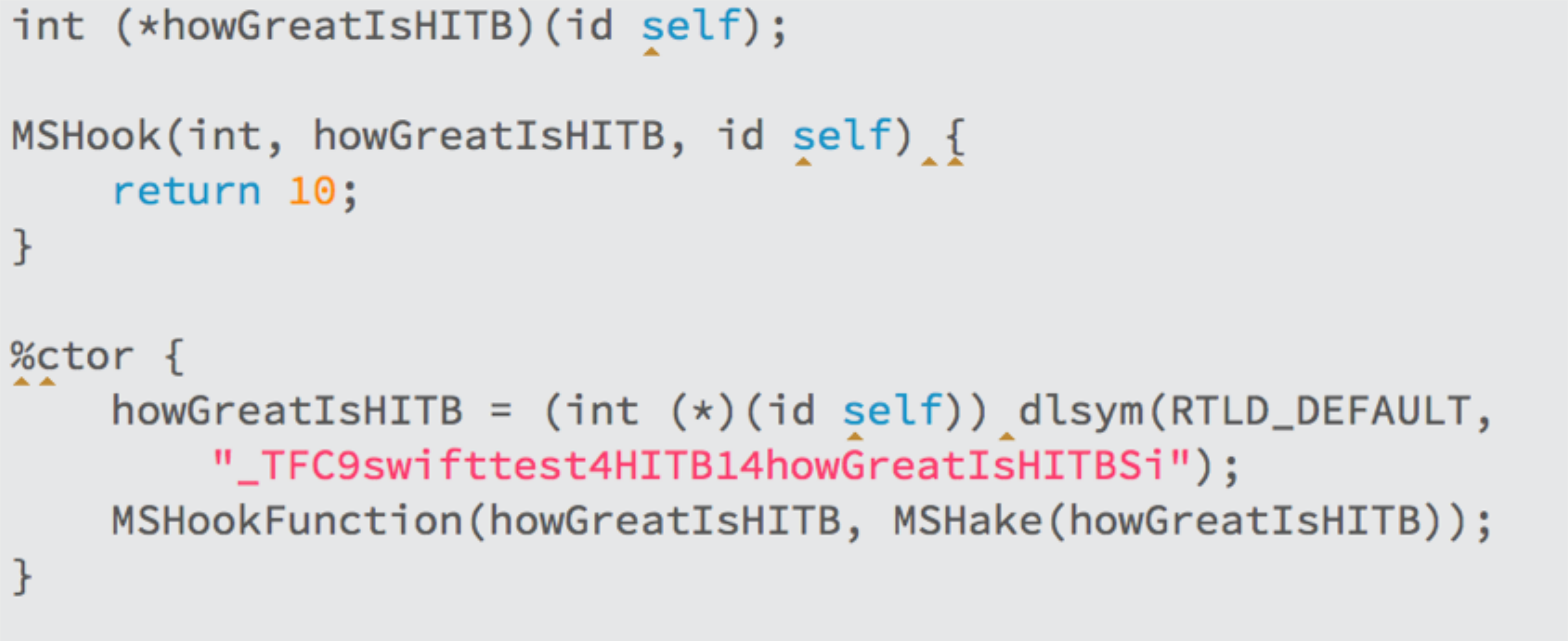
Hooking Swift Methods



```
class HITB {
    var howGreatIsHITB: Int

    init() {
        howGreatIsHITB = 5
    }
}
```

Hooking Swift Methods

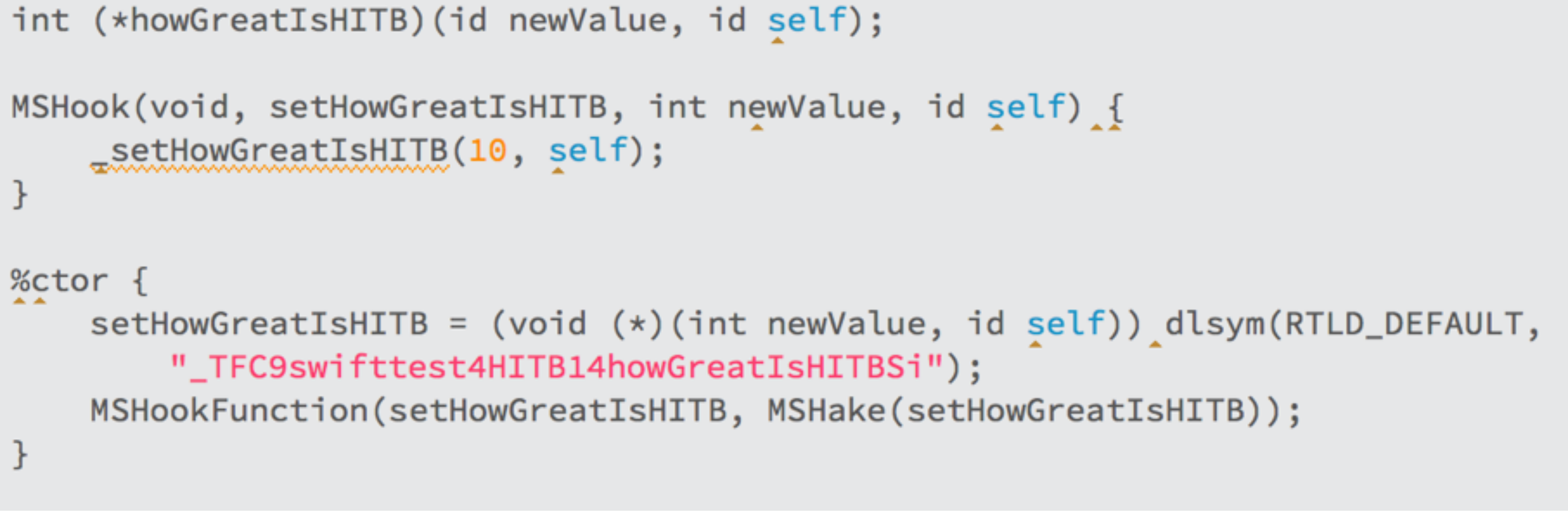
- Hooking getter method (works!)



```
int (*howGreatIsHITB)(id self);

MSHook(int, howGreatIsHITB, id self) {
    return 10;
}

%ctor {
    howGreatIsHITB = (int (*)(id self)) dlsym(RTLD_DEFAULT,
        "_TFC9swifttest4HITB14howGreatIsHITBSi");
    MSHookFunction(howGreatIsHITB, MSHake(howGreatIsHITB));
}
```

Hooking Swift Methods

- Hooking setter method (kinda works…)



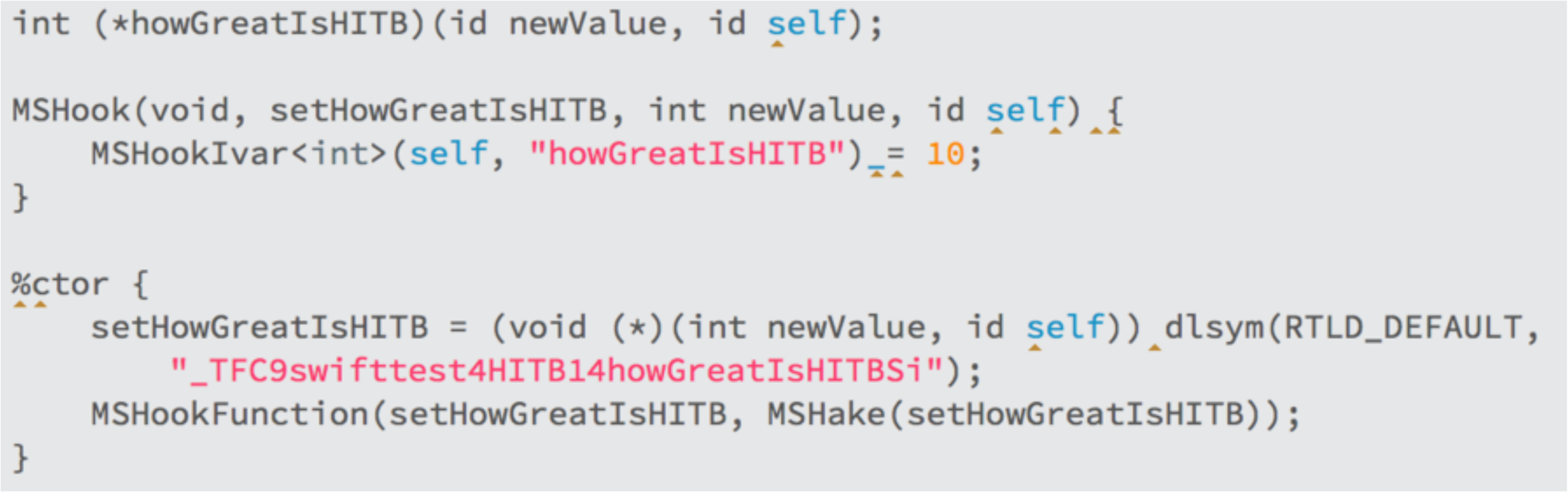
```
int (*howGreatIsHITB)(id newValue, id self);

MSHook(void, setHowGreatIsHITB, int newValue, id self) {
    _setHowGreatIsHITB(10, self);
}

%ctor {
    setHowGreatIsHITB = (void (*)(int newValue, id self)) dlsym(RTLD_DEFAULT,
        "_TFC9swifttest4HITB14howGreatIsHITBSi");
    MSHookFunction(setHowGreatIsHITB, MSHake(setHowGreatIsHITB));
}
```

Hooking Swift Methods

- Certain functions in Swift are inlined and the class constructor is one of them (which is directly setting the instance variable)
- So in this case the setter will only be called again by the top level code.
- If you call from there it works.

Hooking Swift Methods

- Changing the instance variable directly (works but not a good idea probably)



```
int (*howGreatIsHITB)(id newValue, id self);

MSHook(void, setHowGreatIsHITB, int newValue, id self) {
    MSHookIvar<int>(self, "howGreatIsHITB") = 10;
}

%ctor {
    setHowGreatIsHITB = (void (*)(int newValue, id self)) dlsym(RTLD_DEFAULT,
        "_TFC9swifttest4HITB14howGreatIsHITBSi");
    MSHookFunction(setHowGreatIsHITB, MSHake(setHowGreatIsHITB));
}
```

Wrap Up

Wrap Up

- So not all hope is lost when it comes to your standard pen test workflows with Swift apps
- A bit more of a pain in the arse if you don't get access to the source code
- Most issues in iOS and OS X apps are due to poor design decisions or misconfiguration and incorrect implementation of Apple and third party frameworks and libraries.

Next Steps

- Improve the class-dump-s script :)
- Runtime inspection (was going to demo this but ran out of time :( )
    - cycript works but not as straightforward as with Objective-C
    - LLDB works well if you are familiar with it
    - Will hopefully write a blog post soon

Q&A?

Trustwave®
SpiderLabs®