

# 애플 생태계로 들어가는 가장 빠른 길

## 애플 스위프트

현재 개발자 커뮤니티에서 가장 핫한 언어가 스위프트다. 2014년 출시된 이후 1년 만에 프로그래밍 언어 순위 20권에 안착했다. 구글의 '고' 언어가 4년 걸린 것과 비교하면 그 성장세를 짐작할 수 있다. 특히 최근 오픈소스로의 전환은 스위프트에 새로운 날개를 달아줄 것으로 기대된다. 최신 스위프트 2 버전을 중심으로 주요 특징과 활용법을 상세하게 살펴본다. 리눅스와 안드로이드, 윈도우 등 다른 플랫폼으로의 확산 가능성도 점검한다.

- 🔗 지금 스위프트를 시작해야 하는 10가지 이유
- 🔗 애플 생태계의 초대, 스위프트 2
- 🔗 예제 코드로 본 스위프트 2의 7가지 매력
- 🔗 “기획부터 앱 등록까지” iOS 게임 개발 5단계
- 🔗 애플 위치용 앱 개발 입문자를 위한 팀 모음
- 🔗 윈도우 개발자도 주목해야 할 스위프트의 변화

무단 전재 재배포 금지

본 PDF 문서는 IDG Korea의 프리미엄 회원에게 제공하는 문서로, 저작권법의 보호를 받습니다.  
IDG Korea의 허락 없이 PDF 문서를 온라인 사이트 등에 무단 게재, 전재하거나 유포할 수 없습니다.

# 지금 스위프트를 시작해야 하는 10가지 이유

Paul Solt | InfoWorld

**프**로그래밍 언어는 쉽게 죽지 않는다. 그러나 쇠락하는 프로그래밍 언어에 목을 매는 개발자는 그럴 수 있다. 당신이 모바일 앱을 개발하고 있거나 관심이 있는데도 '스위프트(Swift)'를 유심히 보지 않았다면 실수한 것이다. 현재 스위프트는 맥과 아이폰, 아이패드, 애플 워치 그리고 앞으로 출시될 애플 기기용 앱 개발에서 기존 언어인 '오브젝티브-C(Objective-C)'를 대체하고 있다. 결국, 애플 플랫폼 기반의 임베디드 프로그래밍에서 C를 대신하게 될 것이다. 그리고 애플의 전략이 잘 실현된다면 몰입형, 반응형, 소비자 애플리케이션을 개발하는 데 있어 아주 유용한 프로그래밍 언어로 자리 잡을 것이다.

애플은 스위프트를 이용해 매우 큰 목표를 꿈꾸고 있는 것으로 보인다. 스위프트를 이용해 'Hello World'와 같은 초보 프로그래밍부터 완전한 운영시스템까지 개발할 수 있도록 지원할 것이라고 밝혔다. 개발 용이성과 성능을 위해 컴파일러도 최적화했다. 애플은 아직 스위프트의 비전을 모두 공개하지 않았다. 그러나 엑스코드(Xcode), 플레이그라운드(Playgrounds), 스위프트를 보면, 애플이 앱 개발을 더욱 쉽게 하면서도 접근성 높게 만들려는 의도가 있음을 알 수 있다. 지금 당장 스위프트를 시작해야 하는 10가지 이유를 살펴보자.

## 첫 번째 이유. 스위프트는 더 읽기 쉽다

오브젝티브-C는 C에 기반을 둔 언어다. 당연히 C 언어가 가진 불편함을 그대로 품고 있다. 키워드(Keyword)와 타입(type)을 구별하기 위해 사용하는 '@' 부호가 대표적이다. 모든 오브젝티브-C 타입이나 객체 관련 키워드 앞에 @를 붙여야 한다. 반면 스위프트는 C에 기반을 둔 언어가 아니다. 따라서 모든 키워드를 통합하고, 모든 오브젝티브-C 타입과 관련 키워드 앞의 수많은 @ 부호가 필요 없다.

또한, 스위프트에는 구식 규약이 존재하지 않는다. 즉 줄을 끝내기 위해 세미콜론(; )을 쓸 필요가 없다. if/else 명령문 내부의 조건줄을 둘러싸기 위해 괄호를 사용하지 않는다. 이 밖에 메소드 호출(Method call)을 꺾음 괄호( ) 안에 넣지 않아도 된다. 대신 스위프트에서 메소드 호출과 함수 호출은 소괄호 안에 콤마로 분리한 매개변수 리스트를 사용한다. 스위프트는 오브젝티브-C보다 단순한 구문과 문법을 가진 더 명확한 언어다. 스위프트의 특징은 또 있다. 스위프트

코드는 다른 현대적인 프로그래밍 언어를 닮았다. 또 자연어 영어에 더 근접한 언어이다. 이런 가독성 때문에 자바스크립트(JavaScript), 자바(Java), 파이썬(Python), C#, C++ 개발자는 오브젝티브-C와 달리 손쉽게 스위프트를 이용해 프로그래밍할 수 있다.

### 두 번째 이유. 스위프트는 유지 관리가 더 쉽다

오브젝티브-C는 C를 기반으로 하므로 ‘과거에 발목 잡힌’ 언어이다. 즉 C가 개선돼야 오브젝티브-C도 발전할 수 있다. C는 개발자가 앱 개발 시간과 효율성을 개선하기 위해 코드 파일을 2개 유지할 것을 요구하며, 이것은 오브젝티브-C도 마찬가지이다. 반면 스위프트는 이 ‘2개 파일 유지’ 요건을 없앴다. 엑스코드와 LLVM 컴파일러가 종속성을 파악해 자동으로 점진적인 빌드를 수행한다. 그 결과 콘텐츠 테이블(헤더 파일)을 본문(실행 파일)에서 분리하는 반복 작업을 할 필요가 없다. 스위프트는 오브젝티브-C 헤더 파일(.h)과 실행 파일(.m)을 하나의 코드 파일(.swift)로 결합한다.

오브젝티브-C의 2개 파일 시스템은 그동안 개발자의 업무를 늘리는 주범이었다. 더 큰 그림을 보지 못하게 하는 방해물이기도 했다. 운이 좋아야 표준 규약을 이용해 파일 사이의 메소드 이름과 주석을 수동으로 동기화할 수 있었다. 그나마 개발팀 전체가 일정한 규칙과 코드 검토를 꾸준히 하고 있어야 가능했다. 반면 스위프트를 사용하면 엑스코드와 LLVM 컴파일러가 자동으로 이를 처리해 개발자의 작업량을 줄여준다. 번거로운 작업 대신 앱 로직 개발에 더 많은 시간을 투자할 수 있다. 스위프트는 개발자의 반복 작업을 줄여줌으로써, 실제로 중요한 것 즉 앱이 지원해야 할 기능과 주석, 코드의 품질을 높일 수 있도록 한다.

### 세 번째 이유. 스위프트는 더 안전하다

오브젝티브-C에서 흥미로운 부분 하나는 nil(null) 포인터를 중심으로 포인터를 처리하는 방식이다. 오브젝티브-C에서는 nil인 포인터 변수로 메소드를 호출하면 아무런 일도 일어나지 않는다. 코드 줄이나 표식은 no-op이 된다. 충돌하지 않는다는 측면에서는 장점이지만, 버그의 온상이라는 단점이 있다. no-op은 예상하지 못한 행동을 초래할 수 있다. 이는 이상 행동과 충돌 문제를 찾아, 수정하고 방지해야 하는 개발자에게 적과 같은 존재다. 반면 스위프트에서는 타입을 선택할 수 있어 코드의 nil 옵션값을 분명하게 만들어 준다. 나쁜 코드를 쓰면 컴파일러 오류가 발생할 수 있다는 의미이다. 이는 피드백을 받는 주기를 줄여주고 개발자가 의도한 대로 코드를 쓸 수 있도록 도와준다. 코드를 쓰면서 문제를 고칠 수 있다. 결국, 오브젝티브-C 포인터 로직과 관련된 버그를 수정하는 데 낭비될 돈과 시간을 크게 줄일 수 있다.

기존 오브젝티브-C에서는 메소드에서 값을 소환하면, 개발자가 소환된 포인

터 변수의 행동을 주석과 메소드 이름 규약을 이용해 문서화해야 했다. 반면 스위프트에서는 옵션 타입과 값 타입이 값이 존재하거나 옵션이 될 가능성이 있다면, 즉 값이 존재하거나 nil일 경우 메소드 정의에서 명확히 표시된다. 이처럼 스위프트는 예측 가능한 행동을 제공하기 위해 nil 옵션값을 사용하면 런타임 충돌을 촉발한다. 이 충돌이 일관된 행동을 제공하고 버그 수정 프로세스를 더 단순하게 만든다. 개발자가 즉시 문제를 수정하도록 강제하기 때문이다. 특히 스위프트 런타임 실행 오류는 nil 옵션값을 사용한 바로 해당 줄에서 나타난다. 즉 스위프트 코드에서 버그를 더 빨리 찾아 수정하거나 처음부터 방지할 수 있다는 의미이다.

#### 네 번째 이유. 스위프트는 메모리 관리 기능이 통합돼 있다

스위프트는 오브젝티브-C에는 존재하지 않는 방법으로 언어를 통합한다. 절차와 객체 지향 코드 경로 모두에서 ARC(Automatic Reference Counting)를 지원한다. 오브젝티브-C에서는 코코아(Cocoa) API와 객체 지향 코드에서 ARC를 지원했지만, 절차형 C 코드와 코어 그래픽스 같은 API에서는 사용할 수 없었다. 이 때문에 코어 그래픽스 API와 다른 iOS용 저수준 API를 작업할 때는 개발자가 직접 메모리를 관리해야 했다. 이 과정이 매끄럽지 못하면 대규모 메모리 누수가 일어날 수 있었다. 이 부분은 오브젝티브-C가 갖고 있는 가장 큰 문제점 중 하나였다.

반면 스위프트를 이용하면 개발자는 자신이 만든 모든 디지털 객체에 대해 메모리를 생각할 필요가 없다. ARC가 컴파일 시간에 모든 메모리 관리를 처리하기 때문이다. 개발자는 중요한 앱 로직과 새로운 기능에 더 큰 초점을 맞출 수 있다. 스위프트의 ARC는 절차형 코드, 객체 지향형 코드를 모두 지원하므로 개발자는 저수준 API를 쓸 때도 혼동을 피할 수 있다. 애플은 스위프트에 자동 메모리 관리, 고성능 메모리 관리 기능을 추가함으로써 생산성을 높일 수 있음을 입증했다. 다른 부수 효과도 있다. 오브젝티브-C와 스위프트 모두 자바와 고우(Go), C#같이 미사용 메모리를 반환하는 ‘가비지 컬렉터(Garbage Collector)’ 문제가 없다. 아이폰과 애플 워치, 아이패드 등 터치 장치용 반응형 그래픽과 사용자 입력을 사용하는 프로그래밍 언어에서는 매우 중요한 요소이다.

#### 다섯 번째 이유. 스위프트는 필요한 코드의 수가 적다

스위프트는 오브젝티브-C보다 반목 구문과 스트링 조정에 필요한 코드의 양이 적다. 오브젝티브-C의 텍스트 스트링 작업은 매우 복잡하고, 두 종류의 정보를 결합하는 데 많은 단계가 필요했다. 반면 스위프트에는 현대 프로그래밍 언어의 장점이 그대로 구현됐다. 0+1 오퍼레이터로 2개 스트링을 함께 추가하는 것을 예로 들 수 있다. 오브젝티브-C에서는 지원하지 않는 기능이다. 또한, 스위프트는 캐릭터와 문자의 결합을 지원한다. 이는 텍스트를 스크린에서 표시

하는 모든 프로그래밍 언어의 근간이 되는 기능이다.

스위프트 타입 시스템은 컴파일러가 타입을 식별할 수 있어 코드 명령문의 복잡성을 줄인다. 오브젝티브-C는 개발자가 %s, %d, %@ 같은 특수 스트링 토큰을 기억해야 하고 각 토큰을 교체할 변수의 리스트를 콤마로 분리해 제공해야 했다. 반면 스위프트는 스트링 삽입을 지원해 개발자가 토큰을 암기할 필요가 없다. 또 변수를 라벨이나 버튼 제목 등 사용자가 인식하는 스트링으로 직접 삽입할 수 있도록 지원한다. 타입 추론 시스템과 스트링 삽입, 이 두 가지는 그동안 오브젝티브-C에서 자주 발생하는 애플리케이션 충돌의 원인이었다. 순서가 뒤섞이거나 잘못된 스트링 토큰을 이용하면 앱이 충돌을 일으키곤 했다. 스위프트는 이를 해결했을 뿐만 아니라 유지 관리 작업도 크게 줄였다. 작성해야 할 코드 양이 적기 때문이다. 물론 이 때문에 오류 가능성도 줄어들었다.

### 여섯 번째 이유. 스위프트는 더 빠르다

스위프트는 구식 C 규약의 족쇄에서 벗어나 성능이 크게 개선됐다. 스위프트 코드 성능에 대한 벤치마크 결과를 보면 애플이 스위프트의 앱 로직 실행 속도를 높이는 데 꾸준히 노력하고 있음을 알 수 있다. 유명 성능 벤치마크 업체인 프라이메이트 랩스(Primate Labs)에 따르면, 2014년 12월 만델브로(Mandelbrot) 알고리즘을 이용한 컴퓨팅 처리 속도 테스트에서 스위프트의 성능은 거의 C++에 근접한 것으로 나타났다.

2015년 2월에 나온 엑스코드 6.3 베타 버전에서 애플은 GEMM 알고리즘(순차적인 라지 어레이 액세스 및 메모리 바운드 알고리즘) 성능을 1.4배 개선했다. FFT 실행(라지 어레이에 랜덤 액세스를 하는 메모리 바운드 알고리즘) 성능이 2.6배나 높아졌다. 다른 언어의 훌륭한 점을 흡수하면서 스위프트는 계속해서 성능을 개선해 나갔다. FFT 알고리즘 성능이 8.5배 올리면서 1.1배 성능 개선에 그친 C++를 앞질렀다. 만델브로 알고리즘 성능에서도 C++를 1.03배 미세하게 앞서 나갔다. 이제 스위프트는 FFT와 만델브로 알고리즘 모두에서 C++와 사실상 같은 수준에 도달했다.

### 일곱 번째 이유. 이름 충돌이 적다

오브젝티브-C의 또 다른 문제는 코드 파일명 충돌을 피할 수 있는 네임스페이스를 지원하지 않는다는 점이다(C++는 이를 지원한다). 그래서 오브젝티브-C에서 이름 충돌이 발생하면 링커 오류로 앱이 실행되지 않는다. 해결법이 없지는 않지만, 잠재적인 문제점을 안고 있다. 예를 들어, 페이스북과 당신이 만든 코드를 구분하기 위해 오브젝티브-C 코드에 접두사 2~3자를 넣는 식이다.

반면 스위프트는 암시적인 네임스페이스를 제공한다. 그래서 NSStirng(여기서 NS는 Next Step, 즉 스티브 잡스가 애플에서 해고된 후 재직한 회사 이름이다)이나 CGPoint(CG=Core Graphics) 같은 이름 없이 여러 프로젝트에서 같



은 코드 파일을 이용할 수 있다. 개발자가 오브젝티브-C의 단순 작업을 없애 생산성을 높일 수 있는 기능이다. 오브젝티브-C는 네임스페이스를 지원하지 않아 NSArray, NSDictionary, NSString이라고 써야 했지만, 스위프트에서는 Array, Dictionary, String같이 단순한 이름을 사용할 수 있다.

스위프트의 네임스페이스는 코드 파일이 속한 타깃에 근거를 두고 있다. 개발자가 네임스페이스 식별자를 이용해 클래스나 값을 구분할 수 있다. 오브젝티브-C와 비교하면 매우 큰 변화이다. 오픈소스 프로젝트, 프레임워크, 라이브러리를 자신의 코드

에 통합하는 데 큰 도움이 되기 때문이다. 또한, 여러 소프트웨어 업체가 오픈소스 프로젝트를 통합할 때 충돌을 걱정할 필요 없이 같은 코드 파일명을 생성할 수 있다. 이제 페이스북과 애플은 오류나 빌드 실패 없이 FlyingCar.스위프트라는 객체 코드 파일을 사용할 수 있다.

### 여덟 번째 이유. 스위프트는 동적 라이브러리를 지원한다

아직 많은 사람이 그 진가를 모르고 있지만 스위프트의 가장 큰 변화는 정적 라이브러리에서 동적 라이브러리로 바뀐 것이다(이것은 iOS 8, iOS 7에서 가장 중요한 업데이트이기도 했다). 동적 라이브러리는 앱과 연결하는 실행 코드 집합으로, 현재 만드는 스위프트 앱을 향후 스위프트 언어가 개선된 후에도 새 버전으로 연결해준다. 현재 개발자는 자신이 개발한 앱을 앱스토어에 등록할 때 라이브러리를 함께 제출한다. 둘 다 무결성을 증명하기 위해 개발 인증서로 디지털 서명을 한다. 이것은 스위프트가 iOS보다 더 빨리 진화할 수 있다는 의미로, 현대 프로그래밍 언어에 필요한 부분이다. 라이브러리의 변경 사항을 앱스토어의 앱 최신 업데이트에 모두 포함할 수 있기 때문이다. 더구나 모든 과정이 간단하게 작동한다.

스위프트가 없고 iOS 8이 출시되기 전에는 iOS에서 동적 라이브러리를 지원하지 않았다. 하지만 맥에서는 아주 오랫동안 동적 라이브러리를 지원했다. 동적 라이브러리는 실행 가능한 앱의 외부에 위치하지만, 앱 스토어에서 앱과 함께 내려받는다. 필요한 경우에만 외부 코드를 연동하므로 메모리에 로드되는 앱의 초기 크기를 줄여준다. 이는 모바일 앱이나 애플 위치의 임베디드 앱에서 로딩 시간을 줄여 사용자의 체감 성능을 높이는 효과를 낸다. 이는 iOS 생태계를 다른 모바일 앱스토어와 차별화하는 매우 중요한 장점 중 하나다. 애플은 그동안 에셋과 리소스에만 초점을 맞췄었다. 그러다 이제는 필요에 따라 코드를 컴



파일하거나 링크한다. 이런 접근법은 리소스가 화면에 표시되기까지의 초기 대기 시간을 줄여준다.

스위프트는 동적 라이브러리를 도입해 프로그래밍 언어의 변경과 개선점을 매우 빠르게 배포할 수 있게 됐다. 이제 사용자는 iOS의 정기 업데이트를 기다리지 않고도, 애플이 스위프트에서 개선한 성능과 신뢰성의 혜택을 더 빨리 누릴 수 있게 됐다.

### 아홉 번째 이유. 스위프트 플레이그라운드 는 인터랙티브 코딩을 촉진한다

스위프트에 새로 도입된 플레이그라운드는 숙련된 개발자에게 꽤 유용한 기능이다. 애플에서 일했던 브렛 빅터가 플레이그라운드 기능에 일정 부분 영향을 줬다. 플레이그라운드는 개발자가 완전한 아이폰 앱을 개발하지 않고도 5~20 줄의 코드로 구성된 새로운 그래픽 루틴이나 알고리즘을 테스트할 수 있는 기능이다.

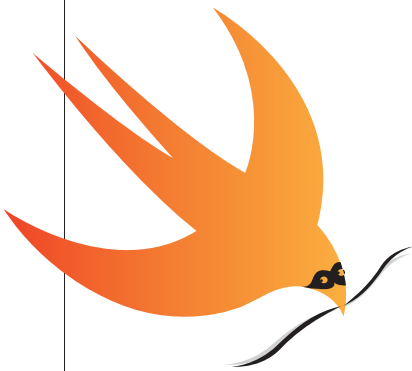
애플은 개발자가 코드나 알고리즘을 개발하는 즉시 피드백을 얻을 수 있도록 플레이그라운드에 인라인 코드 실행 기능을 추가했다. 이 피드백 루프는 코드 개발 속도를 크게 높였다. 전통적인 개발자가 상상 속에서 했던 작업을 시각화해서 보여주기 때문이다. 프로그래밍은 반복 프로세스이다. 따라서 이 기능을 이용하면 개발 과정의 긴장을 줄일 수 있다. 또는 기존의 긴장을 더 창조적인 프로세스를 보완하는 데 돌려서 활용하면 생산성을 높일 수 있다. 또 기존 컴파일러가 개발자에게 강요했던 지루한 세부사항에 초점을 맞추는 대신 더 큰 문제를 해결하는 데 시간을 쓸 수 있도록 한다.

참고로 필자가 초급 개발자를 가르친 경험에 비추어보면, 플레이그라운드는 초급 개발자보다 숙련된 개발자에게 더 도움이 된다. 초급 개발자는 스위프트 플레이그라운드에서 변수의 작동 원리를 확인하는 것만으로는 Floating 포인터 변수와 Integer 변수의 필요성을 이해하기 힘들다. 대신 페이스북 뉴스피드에서 마지막 스크롤 위치를 기억하는 앱을 설명하는 것이 더 빠르다. 초급 개발자에게 '왜'라는 질문에 답을 할 수 있는 것은 아이폰 앱처럼 실제 작동하는 예제뿐이다.

### 열 번째 이유. 스위프트는 개발자가 개선할 수 있다

오브젝티브-C가 사라지는 일은 없을 것이다. 그러나 스위프트가 도입됨에 따라 오브젝티브-C는 크게 바뀌지 않을 전망이다. 스위프트의 일부 기능이 오브젝티브-C에도 도입될 가능성이 있지만, C 언어에 기반을 두고 있어 딱 그만큼만 개선될 것이다. 애플은 개발자가 스위프트에 직접적인 영향을 주는 길을 열어놓았다. 앱과 임베디드 시스템(애플이 써드파티를 대상으로 임베디드 프레임워크와 칩을 라이선스할 경우), 애플 위치 같은 장치에 적합한 개발을 하는 과정에서 말이다.

애플은 최고의 사용자 경험을 제공하는 데 초점을 맞추고 있으며, 주의를 끌




가치가 있는 기능만 구현하고 있다. 애플은 엑스코드 6.3의 스위프트 1.2 릴리스를 통해 애플 버그 리포트(Apple Bug Reporter) 유틸리티를 통해 수집된 수만 개 버그를 수정했다. 스위프트 개발과 발전을 지원하는 팀은 스위프트를 이용해 앱과 시스템을 구현하는 개발자를 더 충실하게 지원할 수 있도록 앱을 개선하는 것에 주력하고 있다.

### 스위프트, 더 접근성 높고 충실한 기능을 갖춘 언어

스위프트는 수많은 개선을 통해 오브젝티브-C를 뛰어넘었다. 오브젝티브-C가 레거시 언어에 뿌리를 두고 있다는 것은 근본적인 한계다. 애플은 애플 특유의 경험을 구현하는 API와 코드 라이브러리인 ‘코코아’를 버릴 계획이 없다. 대신 포스 터치나 탭틱 피드백 같은 기능을 지원하는 새 API를 더 간편하게 사용할 수 있는 기술을 제공하고 있다. 많은 레거시 기술이 컴파일러 디자인의 사용성을 개선하는 데 초점을 맞추고 있다. 스위프트는 기존 레거시 프로그래밍 과정에서 겪었던 정신적인 스트레스를 줄여 앱 개발자가 더 쉽게 작업할 수 있도록 하는 데 집중하고 있다. 다른 현대적인 컴파일러가 발전해 온 것처럼, 더 적은 코드에서 더 많은 정보를 추론할 수 있다.

스위프트를 이용하면 유지 관리해야 할 코드를 절반으로 줄일 수 있다. 또 수동으로 코드를 동기화할 필요가 없으며, 타입 관련 실수도 줄일 수 있다. 이를 통해 코드 품질을 높일 수 있는 시간을 벌 수 있다. 스위프트는 옵션 타입을 추가해 코드를 자동으로 문서화한다. 또 값이 있거나 없는 경우의 호출에 대한 컴파일 타임 안전 메커니즘을 도입했다. 그동안 비동기 작업, 네트워크 실패, 무효한 사용자 입력, 데이터 검증 오류와 관련해 많이 문제가 됐던 부분이다. 스위프트는 애플의 코코아 프레임워크를 이용해 절차형 C 코드와 객체 지향형 코드 모두에 있어 ARC(Automatic Reference Counting)를 통합한다.

개발자가 써야 할 코드가 줄어든다. 또 현대적인 언어의 특징이 코드의 가독성을 높게 유지한다. 스위프트는 계속 발전하면서 애플의 전체 생태계를 풍부하게 할 것으로 보인다. 이 과정에서 iOS와 스위프트에서 동적 라이브러리를 지원하는 것이 큰 도움이 될 것이다. 빌드 타임을 증가시키지 않고도, 오픈 소스 프로젝트, 써드파티 SDK, 홈 오토메이션, 장치, 소셜 서비스와 결합한 프레임워크를 더 손쉽게 통합할 수 있다. 스위프트는 일부 알고리즘에서 C++와 비슷한 수준을 자랑한다. 최근 출시된 엑스코드와 스위프트를 보면 성능이 개선됐음을 알 수 있다. 여기에 플레이그라운드에는 인라인 데이터 시각화를 이용한 알고리즘 개발을 지원하며, 새로운 시각적 피드백을 제공한다. 짧아진 피드백 순환고리와 그래픽 표현이 반복 코딩 프로세스를 더 손쉽게 시작할 수 있도록 도와준다. 결론적으로 스위프트는 앞으로 몇 년 동안 개발자가 앱을 개발하고 애플 워치 같은 새로운 저전력 장치의 임베디드 시스템을 대상으로 개발할 수 있는 더 접근성 높고, 완성도 높은 프로그래밍 언어이다. 



# 애플 생태계로의 초대, **스위프트 2**

Lou Hattersley | Macworld UK

**애플**은 지난해 WWDC 2015 행사에서 스위프트 2(Swift2)를 공개했다. 1년 만의 메이저 업그레이드다. 스위프트는 맥 OS X과 iOS 기기용 앱을 개발할 때 사용하는 언어다. 애플은 기기의 성능을 최대한 끌어 쓸 수 있도록 스위프트를 설계했고, 스위프트 2에서 여러 가지 인상적인 기능을 추가했다. 지난해 말에는 스위프트를 오픈소스로 전환했다. 이에 대해 애플은 “전체 소프트웨어 산업을 진전시킬 기회가 될 것이고 우리가 개발자와 함께 무엇을 만들어갈지 상상으로도 흥분된다”고 밝혔다.

## 스위프트의 오픈소스화가 의미하는 것

일반적으로 오픈소스란 프로그램 소스 코드를 일반에 공개하는 것을 의미한다. 누구나 이를 열람, 수정해 원하는 대로 프로그램을 사용할 수 있다. 애플은 스위프트 소스 코드를 공개했다. LLVM은 C++로 쓰여졌고 기기 하드웨어와 상호작용한다. 이것은 곧 개발자가 비-애플 운영체제에서도 스위프트 프로그래밍 언어를 사용할 수 있다는 의미이다. 이미 리눅스 지원이 진행 중이고 이론적으로 윈도우 컴퓨터에서도 실행할 수 있다. 라즈베리 파이 같은 소형 컴퓨터에서 실행하는 것을 시도하는 개발자도 등장할 것이다.

그러나 OS X과 iOS용 소프트웨어를 윈도우와 안드로이드에서 실행하는 것은 아직 먼 일이다. 스위프트가 오픈소스로 풀리면서 한 기기에서 다른 기기로 소프트웨어를 포트하는 게 더 쉬워졌지만, 애플은 자체 SDK에 대한 제어권은 유지할 가능성이 크다. 맥에서는 여전히 엑스코드(Xcode)를 이용해 OS X과 iOS용 소프트웨어를 개발할 수 있다.

애플이 그동안 철저하게 비밀주의를 유지했고 자사의 제품이 미리 공개되는 것에 대해 매우 민감해 언 듯 보면 오픈소스화는 애플과 어울리지 않아 보인다. 하지만 애플은 오픈소스와 거리를 두는 회사가 아니다. OS X는 유닉스를 기반으로 개발됐고 애플 소프트웨어의 상당 부분이 오픈소스다. 애플의 오픈소스 안내 웹페이지(<https://goo.gl/yXtkrA>)를 보면 다음과 같은 내용이 포함돼 있다. “오픈소스의 코드 요소는 수십 년간 동료 개발자의 검토를 거쳤다. 애플은 오픈소스 활용이 맥 OS X를 더욱 강력하고 안전하게 만든다고 믿는다”

## 스위프트 2의 신기능

스위프트 2에는 개발자를 위한 다양한 신기능이 추가됐다. 다음과 같은 것이 대표적이다.

- **오류 처리 모델.** 스위프트에는 오류를 잡아내고 관리하는 새로운 루틴이 있다. 이를 통해 'file-not-found'나 네트워크 타임아웃 같은 복구 가능한 오류를 처리한다.
- **맥락 개선.** 스위프트에는 코드에 대한 가독성을 높인 맥락 기능이 있다. 애플은 이를 통해 언어의 일관성이 향상될 것으로 기대한다.
- **가용성.** 최신 SDK를 이용하면 새로운 기능을 이용하고 플랫폼에서 개선된 점을 확인할 수 있다. 구형 운영체제에 맞게 컴파일할 수 있고, 이 과정에서 운영체제가 지원하지 않는 API를 사용했을 경우 오류를 표시한다. 또한, 코드를 래핑해 탈옥 등 비정상적인 운영체제에서 실행되지 않도록 할 수도 있다.
- **프로토콜 확대.** 이제 특정 프로토콜을 따르는 모든 클래스에 메소드와 속성을 추가할 수 있다. 이를 통해 개발자가 더 많은 코드를 재활용할 수 있게 됐다.



애플은 오브젝티브-C도 업데이트했다. 오브젝티브-C와 스위프트가 함께 더 잘 작동될 수 있는 신기능이 추가됐다. 제네릭스(Generics), 널러빌리티(Nullability) 주석, 그리고 'Kind-Of'로 알려진 새로운 유형의 객체가 그 예다. 이런 기능은 스위프트와 오브젝티브-C 간 상호작용을 강화한다. 스위프트 2 프로그래밍 언어는 엑스코드 7에서 사용할 수 있다. 맥 앱 스토어 혹은 애플 엑스코드 웹페이지(<https://goo.gl/IOSjH>)에서 내려받을 수 있다.

## 스위프트 프로그래밍 맛보기

스위프트는 맥 OS X과 iOS 기기용 앱을 프로그래밍하기 훨씬 쉽게 설계됐다. 먼저 프로그래밍 언어로서 애플 스위프트에 대해 기본적인 내용을 살펴보고 넘어가자. 스위프트는 점진적으로 기존의 오브젝티브-C 프로그래밍 언어를 대체할 언어다. 오브젝티브-C는 1980년대 개발됐고 지난 1996년 맥에 인수됐다. 2014년까지 17년간 애플은 다른 새로운 프로그래밍 언어를 내놓지 않았으므로, 스위프트는 애플 개발자 커뮤니티에 매우 중요한 언어다.

스위프트는 맥 OS X과 iOS의 프로그래밍 프레임워크인 코코아(Cocoa)와 코코아 터치(Cocoa Touch)용 프로그래밍 언어다. 스위프트 프로그램은 애플의 통합개발환경(IDE)인 엑스코드를 사용해 만들 수 있다. 스위프트는 2014년 도

입된 비교적 새로운 프로그래밍 언어이지만 기존 OS X와 iOS용 개발 언어인 오브젝티브-C 프로그램과 함께 사용할 수 있도록 설계됐다. 이를 통해 개발자는 기존 오브젝티브-C 코드를 바꾸지 않고도 기존 앱에 스위프트 코드를 추가할 수 있다.

스위프트는 오브젝티브-C보다 여러 가지 면에서 장점을 갖고 있다. 무엇보다 스위프트는 구문이 단순하다. 애플 스위프트는 읽고 코딩하기 쉬운 프로그래밍 언어다. 각 줄 마지막에 세미콜론을 넣을 필요가 없고 기능을 이해하기 쉽다. 예를 들어 인쇄 명령은 오브젝티브-C에서 'NSLOG'지만 스위프트에서는 훨씬 익숙한 'println'이다. 코딩하는 데 필요한 특수문자 수도 줄었다. 스위프트는 전반적으로 오브젝티브-C보다 효율적이다.

프로그래밍의 시작인 '헬로 월드(Hello World)' 프로그램(스크린에 'Hello, World!'라고 보여준다)을 만들어 두 언어를 비교해 보자. 먼저 오브젝티브-C의 구문은 다음과 같다.

```
#import
#import <Foundation/Foundation.h>
int main(void)
{
    NSLog(@"Hello, world!\n");
    return 0;
}
```

반면 스위프트에서는 다음과 같이 단 한 줄이다.

```
println("Hello, world!")
```

이처럼 스위프트는 읽고 배우기 훨씬 더 깔끔하고 단순하다. 이 밖에도 스위프트는 다음과 같은 표현식을 지원한다.

- 함수 포인터와 결합한 클로저
- 튜플과 다수의 반환값
- 제네릭스
- 범위와 컬렉션에 걸친 빠르고 간결한 반복
- 메소드, 확장, 프로토콜을 지원하는 구조체
- 지도와 필터 등 기능적 프로그래밍 패턴

두 번째 장점은 메모리를 관리한다는 것이다. 스위프트를 이용하면 개발자가 메모리 할당을 관리하지 않아도 된다. 스위프트 변수는 사용 전 초기화되고,



배열과 정수는 오버플로가 안 되게 검사되며, 메모리는 자동으로 관리된다. 이는 초보 개발자가 더 안전하게 스위프트 프로그래밍 언어를 사용할 수 있게 해준다. 특히 메모리 관리는 앱을 더 안정적으로 만들어 앱 개발자와 앱 사용자 모두에게 도움이 된다. 스위프트를 사용할 수 있는 엑스코드는 맥 앱스토어에서 무료로 내려받을 수 있다.

### 스위프트 프로그래밍을 공부하는 방법

개발자의 대체적 의견은 스위프트가 훌륭한 프로그래밍 언어라는 것이다. 스위프트는 역시 높은 평가를 받는 프로그래밍 언어 ‘파이썬’과 여러 면에서 유사하다. 스위프트는 오브젝티브-C보다 새로운 개발자 관점에서 공부하기 훨씬 쉽다. 파이썬은 여러 프로그래밍 강좌가 기본적으로 가르치므로 많은 개발자가 애플 스위프트 구문에 친숙함을 느낄 것이다.

경력 많은 노련한 개발자에게도 스위프트의 깔끔한 구문은 더 쓰기 쉽게 느껴질 것이다. 특히 오브젝티브-C를 오랜 기간 공부해 왔다면 스위프트가 얼마나 편한지 한눈에 알아볼 수 있다. 물론 새로운 코드를 배우기는 쉽지 않다. 그러나 스위프트 코딩에 익숙해지는 것은 경력 많은 오브젝티브-C 개발자에게도 장기적으로 더 이득이 될 것이다. 현재는 애플이 오브젝티브-C와 스위프트를 모두 코코아와 코코아 터치 앱 개발에 사용할 수 있도록 지원하고 있지만, 점차 스위프트로 옮겨갈 것이 확실시되기 때문이다.


스위프트에 관심이 생겼다면 이제 스위프트의 세계로 안내해 줄 여러 리소스를 살펴보자. 가장 먼저 살펴봐야 할 것은 애플이 만든 스위프트 책 ‘더 스위프트 프로그래밍 랭귀지(The Swift Programming Language)’(<https://goo.gl/J3HR9S>)이다. 애플 개발자 등록 없이도 아이튠즈에서 내려받을 수 있다. 이 책은 기본 연산자부터 상속까지 상세하게 다룬다. 반면 엑스코드 환경에서의 iOS 앱 개발 방법은 수록돼 있지 않다. 조만간 더 자세한 안내서가 나오기를 기대한다. 다음은 스위프트 프로그래밍 관련 참고자료와 링크다.

- The Swift Programming Language (iBooks Store) (영문) (<https://goo.gl/J3HR9S>)
- The Swift Programming Language (영문) (<https://goo.gl/GfGUjv>)
- Using Swift with Cocoa and Objective-C (영문) (<https://goo.gl/LbO3ch>)

유데미(Udemy) 같은 온라인 학습사이트도 추천한다. 다음과 같은 스위프트 관련 강좌가 개설돼 있다.

- iOS 9 and Swift Mastery: Build 11 apps with Swift (<https://goo.gl/nws651>)
- Complete iOS 9 developer course (<https://goo.gl/o3qFs3>)
- Learn to build 20 websites and 14 iOS 9 apps with Swift (<https://goo.gl/rVLUO5>)

프로그래밍을 처음 시작하거나 개론적인 정도로 프로그래밍을 배우고 싶다면 다음 사이트를 확인하자.

- 코드아카데미(<https://www.codecademy.com>). 이 무료 온라인 학습 커뮤니티는 디지털 기술을 가르친다. 아직 스위프트 프로그래밍 과정은 없지만, 파이썬을 배우는 게 도움이 될 것이다. 파이썬은 처음 시작하기 매우 좋은 언어다.
- MITX 6.00.1x: 파이썬을 사용한 컴퓨터공학과 프로그래밍 소개(<https://goo.gl/kWe8IW>). 이 강의는 컴퓨터나 프로그래밍에 경험이 없는 MIT와 하버드 학생을 위해 제작됐다. 모든 자료와 강의를 온라인으로 이용할 수 있다. 프로그래밍을 처음 시작할 때 가장 좋은 가이드라고 할 수 있다
- 런 파이썬 더 하드 웨이(<http://learnpythonthehardway.org>). 사이트명은 비장하지만 내용은 프로그래밍 초보자에게 안정맞춤이다. 

# ITWORLD

테크놀로지 및 비즈니스 의사 결정을 위한 최적의 미디어 파트너



## 기업 IT 책임자를 위한 글로벌 IT 트렌드와 깊이 있는 정보

ITWorld의 주 독자층인 기업 IT 책임자들이 원하는 정보는 보다 효과적으로 IT 환경을 구축하고 IT 서비스를 제공하여 기업의 비즈니스 경쟁력을 높일 수 있는 실질적인 정보입니다.

ITWorld는 단편적인 뉴스를 전달하는 데 그치지 않고 업계 전문가들의 분석과 실제 사용자들의 평가를 기반으로 한 깊이 있는 정보를 전달하는 데 주력하고 있습니다. 이를 위해 다양한 설문조사와 사례 분석을 진행하고 있으며, 실무에 활용할 수 있고 자료로서의 가치가 있는 내용과 형식을 지향하고 있습니다.

특히 IDG의 글로벌 네트워크를 통해 확보된 방대한 정보와 전세계 IT 리더들의 경험 및 의견을 통해 글로벌 IT의 표준 패러다임을 제시하고자 합니다.

# 예제 코드로 본 스위프트 2의 7가지 매력

Paul Solt | InfoWorld

애플이 2014년 6월에 스위프트(Swift)를 처음 발표했을 때, 많은 전문가가 성능, 접근성, 용이성 부분에서 좋은 평가를 내렸다. 그리고 1년가량 지나 스위프트 2를 발표하면서, 코드 가독성과 유지보수 용이성을 한층 더 신속하게 끌어올렸다. 이제는 개발 부서와 IT 부서가 사용하고 기대할 수 있는 수준이다. 스위프트 2는 앞으로 몇 년 동안 계속 쉽게, 그리고 안전하게 유지할 수 있는 언어가 될 전망이다. 특히 오픈 소스에 기반을 두고 있다는 점을 고려하면 더 그렇다.

스위프트 2의 새 언어 구성체(Keywords)는 코드 줄의 실행 순서인 제어 흐름의 가독성을 높인다. 이 새로운 키워드 덕분에 스위프트 코드의 협업 생산성과 효율성이 높아질 것으로 기대된다. 애플은 또 엑스코드(Xcode)를 강화했으며 OS X 10.11, iOS 9, 워치OS 2의 SDK를 개선했다. API는 가벼운 제네릭과 OptionSetType 등 오브젝티브-C용 신기능이 도입되는 방향으로 개선됐다. 이는 스위프트 코드에서 호출할 때, 오브젝티브-C API를 스위프트와 유사하게 만든다.

기업 개발팀은 특히 비즈니스 측면에서 쉽게 적용할 수 있는 코드를 선호한다. 복잡한 코드는 유지 관리하기 힘들고, 심각한 버그를 초래하며, 보안 위협으로 이어질 수 있기 때문이다. 여기 개발자가 더 효율적으로 협력하고, 코드 베이스를 안전하게 만들 수 있는 스위프트 2의 신기능 7가지를 소개한다.

## guard 키워드

스위프트 2의 새 guard 키워드는 오브젝티브-C의 if/else 명령문과 유사한 전제조건 검사 기능을 제공한다. 이는 두 가지 장점을 갖고 있다. 첫째, guard 키워드를 사용하면, 프로그래머가 특정 전제조건이 그 뒤를 따르는 코드 실행에 적합한지 확인할 수 있다. 둘째, 제어 흐름을 판독하기 쉽다. 끝과 시작이 명확하지 않은 수많은 if/else 구조를 집어넣지 않아도, 한 곳에서 간결하게 코드를 확인, 정리할 수 있다. 코드 유지 관리성 측면에서 큰 장점이다. 다음 예제를 보자. guard 명령문 다음에 나오는 코드가 작업 수행에 필요한 모든 파라미터에 대한 사용 준비가 완료됐음을 알려준다.



```

// A value type struct to store data (passed by value i.e. copied!)
struct EmailSubscriber {
    var firstName: String
    var email: String
}

// A supporting function that illustrates valid/invalid values
func validateEmail(email: String) -> Bool {
    return true// false // Implementation as a reader exercise
}

// Create a new subscriber if the parameters meet the requirements
func createEmailSubscriber(firstName: String, email: String) -> EmailSubscriber? {
    // Prevent empty text input from the user
    guard firstName.characters.count > 0 else {
        print("Invalid First Name")
        return nil
    }

    // Assert it's an accepted email format: i.e. Paul@SuperEasyApps.com
    guard validateEmail(email) else {
        print("Email Format Error: Invalid Email")
        return nil
    }

    // Any code path that reaches this point has been validated
    return EmailSubscriber(firstName: firstName, email: email)
}

// Missing data results in a nil value
let invalid = createEmailSubscriber("", email: "Paul@SuperEasyApps.com")
print("Subscriber is invalid, returned: ", invalid)

// Attempts to force unwrap and use nil values causes crashes!
// Tip: Use the if/let syntax to work safely with optional types
//print("Subscriber is invalid, returned: ", invalid!.firstName) // ERROR!

// Complete data results in a new digital subscriber value/object
let valid = createEmailSubscriber("Paul", email: "Paul@SuperEasyApps.com")
print(valid!.firstName, "'s email is: \(valid!.email)")

```

이는 스위프트의 핵심 강점 중 하나를 더 강력하게 만들어준다. 즉 유효하지 않은 값, 옵션값을 보호하는 메커니즘이다. guard를 이용하면 핵심 앱 논리를 더 간결하게 표시해 유지할 수 있다. 사전에 유효하지 않은 상태를 차단할 수 있기 때문이다.

### defer 키워드

스위프트 2의 새 defer 키워드는 스위프트에 없었던 제어 흐름 수준 하나를 추가시킨다. defer 키워드를 이용하면 손쉽게 값비싼, 또는 제한된 리소스를 정리할 수 있다. 예를 들어, 수만 개 파일을 읽고 쓰는 드롭박스용 앱을 개발한다고 가정하자. 코드가 읽기와 쓰기를 마친 후 파일을 닫는 데 실패하면 앱 프로세스의 파일 설명자(file descriptor)가 최대 수치에 도달할 수 있다. 여기에 defer 명령문을 이용하면 파일 처리가 완료됐을 때, 또는 오류가 발생했을 경우 파일을 닫도록 할 수 있다.

defer의 중요 강점은 코드를 상단과 하단에 분리하지 않고, 함께 유지할 수 있다는 것이다. 클린업 코드(파일 닫기)를 현 유효 범위의 끝에서 실행될 defer 종결 형태로, 값비싼 또는 제한된 자원(파일 열기) 생성 아래 직접 쓸 수 있다. 다음 코드를 보자.

```
func fileProcessing() {
    print("1. Create file descriptor #1 and start file processing")
    defer {
        print("5. Close and cleanup file descriptor #1 (I/O resource)")
    }

    print("2. Create file descriptor #2 and start file processing")
    defer {
        print("4. Close and cleanup file descriptor #2 (I/O resource)")
    }
    print("3. Finish file processing")
}

// Look at Xcode's Console output for order: 1, 2, 3, 4, 5.
fileProcessing()
```

앞의 코드 블록에 메시지를 1, 2, 3, 4, 5의 순서로 엑스코드 콘솔로 프린트하는 코드다. defer 키워드는 defer 구역의 코드가 현재 유효 범위의 종결 괄호 이후 실행되도록 만든다. 여러 defer 코드 블록을 메소드와 기능 내부에 추가할 수 있다. 각 defer 종결은 호출되는 역순으로 실행된다. 이는 스위프트 2의 새 오류 처리 모델과 결합해 안전한 클린업을 보장한다.

### repeat/while 및 do 유효 범위

애플은 최근 스위프트의 제어 흐름에 놀라운 변화를 추가했다. do/while이 이제 repeat/while 루프가 됐다. 작은 변화처럼 보일 수 있다. 그러나 오브젝티브-C, 자바스크립트, C++, 자바 등 기존 언어의 질서와는 다른 방식이다.

repeat 키워드는 코드 블록을 반복하는 인스턴트 큐를 제공하고, 다음과 같이 코드의 가독성을 강화한다.

```
// A method that counts down and outputs to Xcode's Console
func repeatWhile() {
    var x = 10
    repeat {
        print("T-minus", x, "seconds")
        x = x - 1// x-- is being deprecated in Swift 3.0!
    } while x > 0
    print("Blast off!")
}

// Call the method in Playgrounds
repeatWhile()
```

스위프트 2에서 do 키워드는 새 유효범위를 생성할 수 있도록 개선됐다. 이는 이를 루프에 사용하는 다른 유명 언어와 다른 점이다. 기존 유효 범위 내부에 또 다른 내부 유효 범위를 생성하고 싶다면 do를 사용하면 된다. 여유가 없는 루프에서 고급 리소스 할당을 해제할 필요가 있거나, ARC(Automatic Reference Counting)가 효과적으로 값비싼 리소스를 클린업하도록 할 때 유용하다.

```
// Outer scope
let x = 7
do {
    // Inner scope (new x variable masks outer scope x variable)
    let x = 10
    do {
        // Inner inner scope ... inception
        let x = 200
        print("x:", x) // x: 200
    }

    print("x:", x) // x: 10
}
// outer scope
print("x:", x) // x: 7
```

C++나 오브젝티브-C와 마찬가지로 do 유효범위로 로컬 변수의 라이프타임 제어를 강화할 수 있다. 어느 경우에 유용할까? 여유가 없는 루프에서 아주 큰 데이터 파일이나 15메가픽셀의 이미지를 차례로 처리해야 한다면 메모리가 부

족할 수 있다. iOS가 강제로 앱을 종료시키기 때문이다. iOS 메모리에는 제한이 있어 이를 넘어서면 앱이 충돌을 일으킨다. 어렵게 개발한 앱에 대해 사용자가 불만을 느낄 가능성이 크다. 이때 do 유효 범위를 이용하면 이런 충돌을 방지하고 여유가 없는 루프 끝 대신 각 필터가 통과한 이후 ARC를 이용해 메모리를 재할당할 수 있다.

### 오류 처리

앞서 살펴본 새로운 키워드는 스위프트 2의 새 오류 처리 모델과 완벽하게 조화된다. 오류 처리는 스위프트 2의 가장 중요한 개선점 중 하나다. 그전까지 스위프트에는 없었고, 오브젝티브-C에서는 사용하기 값비쌌던 기능이었다. 스위프트의 오류 처리는 네트워킹, 입력, 데이터의 예상하지 못한 문제를 처리하는 함축적인 제어 흐름 구조를 생성한다. 스위프트는 자바에서 자주 쓰이는 기존 try/catch 구조의 유효 범위를 지정할 때 do 키워드를 사용한다. 스위프트에서는 do/catch 블록을 사용한다.

스위프트 2의 오류 처리가 모든 것을 해결하지는 않는다. 그러나 사용자 양식의 유효하지 않은 입력, 웹 서비스의 유효하지 않은(예상하지 못한) 응답 데이터 등 복구 가능한 문제를 해결하는 데 도움이 된다. 휴대전화 신호 수신과 휴대전화의 제한된 디스크 용량 등과 관련된 문제, 네트워크 가용성, 파일 I/O 관련된 문제의 오류 처리에 사용할 수 있다.

오류 처리는 throw 키워드를 이용해 메소드 시그니처(method signature) 내에서 구축된다. 오차 전파(error propagation)를 지원하므로 모든 API에서 중복되는 추가 NSError 포인터 파라미터를 없앨 수 있다. 실패할 수 있는 코드(안전하지 못한 코드) 줄에는 일일이 try 키워드를 표시해야 한다. 이렇게 하면, 코드를 작업하는 개발자가 실패할 수 있는 코드줄임을 한눈에 파악할 수 있다. 실패할 수 있는 코드를 처리하는 방법은 다음과 같이 3가지가 있다.

1. try? 키워드를 이용해 오류를 무시하고 이를 통해 스위프트 2 오류 처리 모델과 옵션 타입 시스템을 통합한다. try?를 이용하면 모든 옵션을 없애고, 옵션값을 오른쪽 값으로 되돌릴 수 있다.
2. do/catch 명령문을 이용해 안전하지 못한 코드의 모든 오류를 처리한다.
3. 메소드나 기능 시그니처에 throw 키워드를 추가해 오류를 전파한다. 자바와 달리 오류가 기본값으로 전파되지 않는다.

스위프트 2의 오류 처리는 오브젝티브-C 오류 모델을 개선한 것이다. 즉, 도큐멘테이션과 주석을 사용하는 방식이다. 스위프트 2의 오류 처리 모델은 안전하지 못한 코드가 코드 베이스에 삽입되지 않도록 방지하는 컴파일 타임 검사를 통해 안전성을 제공한다. 예를 들어, 다음 코드를 보자. 고객이 진짜 이름과 이

메일 주소를 입력했는지 검증하는 것을 알 수 있다.

```
// Create a custom error type by conforming to the ErrorType protocol
enum ValidationError: ErrorType {
    case InvalidName
    case InvalidEmail
    case InvalidAge(age: Int)
}

// Check name against your name policies
func validateName(name: String) throws {
    // Use guard statements to prevent invalid user input
    guard name.characters.count > 0 else {
        throw ValidationError.InvalidName
    }
}

// Process a new customer using required attributes
func onboardNewCustomer(name: String, email: String, age: Int) {
    do {
        print("Started onboarding")
        // You must use the try keyword for any method that can throw an error
        try validateName(name)
        // Exercise: Validate other required attributes (age, email, etc.)

        // Finished processing if no errors
        print("Finished onboarding")
    } catch let error as ValidationError {
        // Using a local variable you can catch all ValidationErrors
        // The local error variable can be handled with a switch
        switch(error) {
            case ValidationError.InvalidName:
                print("Invalid name!")
            case ValidationError.InvalidEmail:
                print("Invalid birthday!")
            case ValidationError.InvalidAge(let age):
                print("Invalid age \(age)!")
        }
    }
    catch { // default error catch
        print("Catch errors here:", error) // error is default name
    }
}
```

```
onboardNewCustomer("", email: "", age: 12223) // Invalid!
onboardNewCustomer("paul", email: "Paul@SuperEasyApps.com", age: 29)
```

새 오류 처리 모델은 코드베이스에 오류가 있을 때 추가 코드 줄을 복구해 실행하는 데 있어 새 defer 키워드와 잘 어울린다. 자바의 finally 키워드와 유사하다.

### 프로토콜 확장

애플의 스위프트 1에는 여러 단점이 있었다. 예를 들어, 문서화가 취약하고 스위프트 표준 라이브러리에서 찾기 힘든 수많은 글로벌 독립 기능(Global free-standing Function)이 대표적이다. 필자를 포함해 더 빠른 코딩을 위해 코드 완성을 이용하는 경력 많은 오브젝티브-C 개발자는 이를 대체하는 방법을 찾는 데 어려움을 겪었다.

이제 스위프트 2에서는 프로토콜 확장이 새로운 행위(behavior)와 기능과 상태로 자바 인터페이스 등 프로토콜을 강화한다. 이는 스위프트에서 중요한 부분을 차지하고 있으며, 스위프트 표준 라이브러리의 외형과 오픈 소스 코드, 코드 베이스 개발 방식을 바꿔놓고 있다. 프로토콜 확장과 함께 새 API가 도입되면서 대부분의 독립 글로벌 기능이 사라졌다. 그 결과 누구나 표준 닷 구문인 'variableName.methodName()'을 이용할 수 있게 됐다. API를 더 쉽게 발견할 수 있게 됐고, 개발자의 생산성이 높아졌다.

스위프트 2의 프로토콜 확장의 경우 기능이나 메소드가 존재하지 않으며 이를 직접 쓸 수 있다. 예를 들어, 애플이 새 기능이나 메소드로 제공하는 계층(Array) 구조를 확대할 수 있다. 오브젝티브-C에서는 NSMutableArray에 하나의 코드 줄인 여러 요소를 하나의 코드 줄로 제거하는 'removeObjectsInArray:'라는 메소드가 있었다. 스위프트 2.1에는 하나의 메소드 호출을 이용해 스위프트 계층에서 여러 요소를 제거하는 메소드가 없다. 프로토콜 확장을 이용해 이 메소드를 계층 구조에 추가할 수 있다.

```
// Extend the Array class when Elements are comparable
extension Array where Element: Equatable {
    // Remove a single Element if it is found in the Array
    mutating func removeObject(object: Element) {
        if let index = self.indexOf(object) {
            self.removeAtIndex(index)
        }
    }

    // Remove multiple Elements using the previous method and a loop
```



```

        mutating func removeObjectsWithArray(array: [Element]) {
            for object in array {
                self.removeObject(object)
            }
        }
    }

var students = ["John", "Sue", "Michael", "Chris", "David", "Benjamin"]
var studentsToRemove = ["Michael", "David", "Benjamin"]
print("Students: ", students)

// Remove an array of student names
students.removeObjectsWithArray(studentsToRemove)
print("Students: ", students)

students.removeObject("John")
print("Students: ", students)

```

위의 코드에서 알 수 있듯이 이런 유형의 프로토콜 확장은 같은 논리를 유지하고 싶은 모든 코드 파일에 복사/붙여넣기를 하지 않고, 코드 몇 줄로 재사용할 수 있게 해준다.

### OptionSetType

애플은 스위프트의 접근성을 높이고 있다. 스위프트 2에서는 기존의 비트 마스크 옵션을 없애는 변화를 시도했다. 스위프트 2의 새 OptionSetType은 초보자에게는 어려운 bitwise 연산 대신 세트 논리를 사용한다. 기존의 오브젝티브-C와 C API는 여러 연산을 하나의 값으로 통과시키기 위해 논리적으로 결합하는 작업에 bitwise 연산을 사용했다. 비트 단위의 최적화로, 비트 마스크와 bitwise 연산 원리를 이해해야 했다.

그러나 OptionSetType의 경우 비트 마스크를 100% 이해하지 않아도 앱 스토어에서 판매할 수 있는 앱을 만들 수 있다. 이것은 개발자에게 큰 장점이다. 기존의 프로그래밍 모델에서는 상투적으로 반복되는 코드가 있어 가독성이 떨어졌기 때문이다. 애플은 또 새 OptionSetType를 사용하기 위해 NS\_OPTIONS 값이 필요했던 오브젝티브-C API를 모두 쇠신했다. 이로 인해 파괴적으로 코드가 바뀌었지만 스위프트 구문을 지원하는 API와 함께 계속 발전시키고 있다. OptionSetType은 여러 설정을 하나의 파라미터로 손쉽게 이전할 수 있다. 다음 코드에서 볼 수 있는 것처럼 바이너리 연산의 전문가가 될 필요도 없다.

```

// You can create custom flags or options using OptionSetType
struct VideoFormat: OptionSetType {

```

```

let rawValue: Int
// Use static let values within the struct and assign
// the raw value using bitmask values (1, 2, 4, 8, 16, etc)
static let Video1080p = VideoFormat(rawValue: 1)
static let Video720p = VideoFormat(rawValue: 2)
static let EnableStreaming = VideoFormat(rawValue: 4)
static let EnableDownloads = VideoFormat(rawValue: 8)
}

// Use Swift Set notation to pass options
func startVideoPlaybackWithOptions(videoStreamOptions: [VideoFormat]) {
    print("Play video stream:", videoStreamOptions)
}

// Store a collection of options using Set notation
let videoOptions = [VideoFormat.Video1080p, VideoFormat.EnableDownloads]
startVideoPlaybackWithOptions(videoOptions)

// Set notation makes legacy bitmask operations much easier to read
if videoOptions.contains(VideoFormat.Video1080p) {
    print("Video is 1080p!")
}

```

### SDK 현대화와 오브젝티브-C 강화

애플이 스위프트를 새로 내놓음에 따라 오브젝티브-C를 발전시킬지에 대해 논란이 있었지만, 애플은 결국, 스위프트 2에서 오브젝티브-C를 계속 지원할 것임을 분명히 했다. 방식은 스위프트에서 애플 코코아 터치와 코코아 SDK 등 오브젝티브-C API를 호출하는 코드를 직접 쓸 수 있도록 지원해 결과적으로 오브젝티브-C를 강화한 것이다.

애플은 또 컬렉션 형식이 각 요소에 대한 추가 형식 정보를 저장할 수 있도록 오브젝티브-C에 가벼운 제네릭을 추가했다. 이는 오브젝티브-C에 더 많은 형식 안전성을 제공한다. 동시에 컬렉션 형식을 사용하는 API는 스위프트 코드에서 액세스한 같은 형식 안전성을 통과할 수 있음을 의미한다. 이제 스위프트 UIResponder 프로토콜의 touchesBegan(\_:withEvent:) 메소드가 형식 컬렉션을 지원하며 스위프트 2를 쇄신하고 있다. 애플이 스위프트 1.1에서 1.2, 2.1로 발전하는 동안 구문과 API가 어떻게 진화했는지 보면 단적으로 알 수 있다. 애플은 오류 초기화(fallible initializers), 스위프트 세트 컬렉션 형식(Set collection type), 오브젝티브-C의 경량 제네릭 등을 추가했다.

```

class ViewController : UIViewController {
    // Swift 1.1 (APIs used Objective-C NSMutable type)

```

```

        override func touchesBegan(touches: NSSet, withEvent event: UIEvent) {
            if let touch = touches.anyObject() as UITouch? {
                let location = touch.locationInView(self.view)
                println("Touch: \(location)")
            }
        }
    // Swift 1.2 (Swift Set introduced along with new as? keyword)
    override func touchesBegan(touches: Set<NSObject>, withEvent event: UIEvent)
    {
        if let touch = touches.firstas? UITouch {
            let location = touch.locationInView(self.view)
            println("Touch: \(location)")
        }
    }

    // Swift 2.1 (Objective-C gains lightweight generics + Swift API update)
    override func touchesBegan(touches: Set<UITouch>, withEvent event: UIEvent?)
    {
        if let touch = touches.first {
            let location = touch.locationInView(self.view)
            print("Touch: \(location)") // New print() method
        }
    }
}

```

스위프트의 단단한 형식 시스템은 오류가 런타임 충돌(초기에 수정하는 게 좋은 버그)을 유발하기 전에 이를 방지하는 컴파일-타임 검사로 형식 안전성을 강화했다. 컴파일-타임 오류 검사가 중요한 이유가 여기에 있다. 또 컴파일 타임 확인을 대상으로 한 API 가용성 검사를 강화했다. 엑스코드에서 선택한 iOS 버전 전부를 지원하게 됐다. 가용성은 앱을 통합한 퍼블릭 SDK를 앱 개발자에게 배포해야 하는 모든 회사에 중요하다. 마지막으로 nullability는 오브젝티브-C API가 스위프트 옵션 타입 시스템과 호환되도록 도와준다. 또 불러온 오브젝티브-C API를 옵션 타입 정보를 이용하는 스위프트처럼 만든다.


이번 글에서 소개한 예제의 스위프트 플레이그라운드 파일은 [여기](http://goo.gl/T6DLnY)(<http://goo.gl/T6DLnY>)에서 내려받을 수 있다.

### 다음번 앱을 스위프트로 개발해야 하는 이유

애플은 스위프트 2에서 기존 코드를 파괴했지만 중요한 도약을 이뤘다. 스위프트는 계속 이런 식으로 성장하고 있으며, 이것은 개발자 세계의 기준으로 자리 잡는 데 매우 중요하다. 스위프트는 2015년 12월 오픈 소스로 개방됐다. 현재 애플은 스위프트 3.0 이후의 방향에 대해 개발자와 공개 토론을 하고 있다.

스위프트 2의 새로운 제어 흐름 키워드와 언어 구조는 처음 쓰는 코드 줄에서 개발자의 의도를 더 명확하게 하도록 도와준다. 이런 직관적인 방식 덕분에 코드 블록에서 일어나는 일을 알 수 있고, 앱의 제어 흐름을 적절히 유지하는 능력을 높일 수 있다. guard 키워드는 조기 탈출(early exits)을 지원하고, 유효하지 않은 입력으로부터 코드 논리를 보호한다. defer 키워드는 완료 또는 오류 발생 시 리소스를 정리할 수 있도록 도와준다.

새로운 이름이 붙여진 repeat/while 루프와 do 유효범위 지정은 가독성과 개발자의 의도에 기반을 둔 애플의 새 오류 처리 모델을 지원한다. 안전하지 않을 수 있는 메소드 시그니처는 새 throw 키워드로 표시한다. 안전하지 않은 메소드(오류를 초래할 수 있는) 호출은 새 try 키워드로 표시한다. return 명령문과 같으며 오브젝티브-C와 다른 오류 처리는 기준에 부합한다. 여러분이 다음 앱 개발 프로젝트에서 스위프트를 사용하는 오류 처리를 도입해야 하는 이유 중 하나이다.

애플은 제어 흐름을 바꾼 것은 물론 OS X 버전 10.11, iOS 9, 워치OS 2 SDK를 대상으로 하는 기존 API를 현대화했다. 오브젝티브-C API를 스위프트에서 아무 문제 없이 사용할 수 있는 새 언어 기능을 도입했다. 이는 완전히 새로워진 스위프트 2를 즉시 스위프트 1처럼 사용할 수 있다는 의미이다. 심지어 방대한 오브젝티브-C API와 SDK를 이용할 때도 마찬가지이다. 다음 앱 개발에 스위프트를 이용해야 할 이유가 점점 늘어나고 있다. 



## IT 트렌드 종합 정보센터

# IDG Tech Library

IDG Tech Library는 IDG 글로벌 네트워크를 통해 축적된 전문 정보를 재구성하여 최신 기술의 기본 개념부터 현황, 전략 및 도입 가이드까지 다양한 프리미엄 IT 정보를 제공합니다. Computer World, Info World, CIO, Network World 등의 세계적 IT 유명 매체의 심도 깊은 정보를 무료로 만나보세요

IDG Deep Dive, Tech Focus, Summary, World Update 등의 다양한 콘텐츠를 제공 받을 수 있습니다.



**IDG**  
INTERNATIONAL DATA GROUP

한국IDG(주) 서울시 중구 봉래동 1가 108번지 창화빌딩 4층 100-161 Tel : 02-558-6950 Fax : 02-558-6955  
[www.itworld.co.kr](http://www.itworld.co.kr) [www.twitter.com/ITWorldKR](https://www.twitter.com/ITWorldKR) [www.facebook.com/ITworld.Korea](https://www.facebook.com/ITworld.Korea)

# “기획부터 앱 등록까지” iOS 게임을 만드는 5단계

Lou Hattersley | Macworld UK

애플은 그 어느 때보다 아이폰과 아이패드뿐만 아니라 애플워치용 게임을 손쉽게 개발할 수 있도록 지원하고 있다. 새로운 엑스코드(Xcode) 개발 환경과 스위프트(Swift) 프로그래밍 언어, 완전히 새로워진 메탈(Metal) 덕분에 iOS용 게임 개발을 시작하기에 적기이다. 스위프트는 앱 개발에 새로운 활력을 불어넣었다. 앱 개발자에게 황금기가 도래하고 있다.

게임을 만드는 것이 너무 어려워 보일 수 있지만 생각하기만큼 어렵지는 않다. 인디게임 개발자가 되는 데 참고할 수 있는 다양한 자원이 있다. 특히 아이폰과 아이패드용 앱 개발 관련해서는 언급해야 할 것이 많다. 과거와 달리 아이폰과 아이패드의 모든 기능에 접근할 수 있으며, 특히 애플은 iOS 8부터 개발자에게 아이폰과 아이패드를 더 광범위하게 개방했다.



## iOS용 게임을 만들기 위한 준비

여기서는 iOS용 게임 개발에 관해 이야기하고 있지만 여기에서 추천하는 과정과 자원 대

부분은 일반적인 앱 개발에 관해 어느 정도 알고 있다는 가정하에 서술한다. 하지만 완전 초보자라도 걱정할 필요는 없다. 앱 개발을 시작하는 방법은 물론 앱 개발의 기초 단계부터 다루고 있다. 이 단계를 마치면 게임 개발이 더 친숙하게 느껴질 것이다.

희소식은 iOS용 비디오 게임 개발에 관심 있는 사람들이 꽤 된다는 점이다. 그래서 비디오 게임 개발의 기본을 알려주는 다양한 과정이 개설돼 있다. 다음 유데미(Udemy) 과정을 통해 간단한 iOS용 게임을 개발할 수 있다. 다소 구식 이긴 하지만 스프라이트키트(Sprite Kit, 평면 2D 게임 개발에 사용하는 툴)에 관해 상세하게 배울 수 있다.

– The Complete iOS Game Course – Build a Flappy Bird Clone (<https://goo.gl/h1fGL1>)

3D 게임을 개발하고 싶다면 유니티(Unity) 개발환경이 좋다. 크로스 플랫폼(Cross Platform)이어서 한번 배워두면 안드로이드용 게임에도 손쉽게 적용할 수 있다. 다음은 현재 시장에서 가장 인기 있는 비디오게임 개발자 과정 중 하나다.

- Learn To Code by Making Games – The Complete Unity Developer (<https://goo.gl/hgcUXF>)

다음은 효과적인 게임을 개발하는 방법을 배울 때 좋다. 팩맨(Pac Man)에서 영감을 얻긴 했지만 물리학을 이용한 미로 게임을 개발할 수 있다.

- iOS Maze Games with Swift, Sprite Kit and Designed in Tiled (<https://goo.gl/0uHmGq>)

iOS 게임 개발을 공부할 때는 몇가지 꼭 확인해야 할 자료가 있다.

- **스프라이트빌더**(<http://www.spritebuilder.com>). 게임을 위한 스프라이트(캐릭터)를 만드는 가장 좋은 방법이다. 무료로 제공되는 오픈소스이다.

- **Raywenderlich.com**(<http://www.raywenderlich.com>). 유데미와 마찬가지로 광범위한 개발자 과정을 제공한다. 특히, 스프라이트킷 관련 강좌가 다양하다.

- **애플 개발자 웹사이트**(<https://developer.apple.com>). 애플의 자체 개발자 웹사이트를 적극적으로 활용하자. 시간을 갖고 구석구석 살펴보기 바란다. 특히, 개발자를 위한 게임 센터(<https://developer.apple.com/game-center>)는 꼭 확인하자. 여기에는 게임 개발의 여러 측면을 다루는 광범위한 동영상들이 올라와 있다.

- **깃 허브**(<https://github.com>). 아직 깃 허브에 가입하지 않았다면 바로 계정을 생성하자. 다른 개발자와 팀을 이뤄 협업할 수 있는 프로젝트를 찾아보자.

- **게임 개발스택 익스체인지**(<http://gamedev.stackexchange.com>). 개발자라면 당연히 가입해 두는 것이 좋다. 게임 개발 관련된 질문과 고민을 올릴 수 있으며 꽤 신속하게 답변이 올라온다. 문제를 혼자 안고 있지 말자. 다른 개발자와 함께 해법을 찾자.

### 게임 개발 5단계

자 이제 본격적으로 앱 개발을 시작해 보자. 다음의 단계를 거쳐 바로 게임 개발로 옮겨갈 수 있다.

#### 1단계: ADC(Apple Developer Connection)에 가입한다

우선 ADC에서 멤버센터(Member Center)와 등록(Register)을 클릭한다. 애플 개발자로 등록하면 다양한 자료를 이용할 수 있으며 애플에 iOS 기기를 등록하여 개발중인 앱을 테스트할 수 있다. 자신의 애플 ID로 등록하거나 개발자



계정용 애플 ID를 생성할 수 있다. 기업 개발자인 경우 개발자 계정을 만드는 것을 권장한다.

개발자 등록 비용은 기본적으로 없다. 일단 등록하면 iOS용 다양한 개발자 툴을 이용할 수 있다. 앱 개발과 테스트하는 정도라면 기본적인 등록으로 충분하다. 반면 앱을 판매하려면 연간 69달러를 내고 등록해야 한다. 이렇게 ADM(Apple Developer Membership)이 되면 iOS와 엑스코드 등의 베타 버전 소프트웨어에 접근할 수 있다.

### 2단계: 게임을 개발하기 전에 계획을 세운다

앱 개발에 뛰어들기 전에 앱을 계획하는 것이 중요하다. 앱을 계획하는 방법이 정해져 있지는 않지만, 반드시 기록해두어야 할 것을 추리면 다음과 같다.

- **목표.** 궁극적으로 앱을 개발하는 목적은 무엇인가? 가능한 한 단순해야 한다. 카메라 앱은 사진을 촬영하고 단순한 편집과 공유 기능을 제공한다. 음악 앱은 아이튠즈 매치(iTunes Match)의 음악을 재생한다(또는 아이튠즈 동기화한다). 앱의 기능을 설명하는 하나의 문장을 작성한다.
- **범위.** 앱에 포함된 모든 기능에 대해 결정한다. 앱을 개발하는 중간 과정에서 과도한 아이디어가 추가되지 않도록 주의한다. 시작하기에 앞서 어떤 기능을 넣고 뺄지 결정한다.
- **테마와 전략.** 앱의 테마와 전략은 무엇인가? 행복하고 친숙한 앱인가 아니면 삭막하고 효율적인 비즈니스 툴인가? 자신의 마케팅 전략을 결정하고 앱의 스타일을 여기에 일치시켜야 한다.
- **인터페이스를 그린다.** 각 디스플레이를 그리고 각 버튼의 상호작용을 지도로 작성해야 한다.
- **자산을 모은다.** 애플은 다양한 기본 버튼과 아이콘(사용하는 것이 좋다)을 제공하지만 로고, 그래픽, 오디오 자산 등이 더 필요하다.

이제 필요한 기술에 대해 생각해 보자. 엑스코드와 앱 스토어(App Store)에서 그럭저럭 해 나갈 수 있을까? 아니면 앱에 서버와 다른 기술이 필요할까? 개발 단계부터 이 문제를 생각해야 한다. 이 단계에서 참고할 만한 자료는 다음과 같다.

- Mozilla: Planning your app (<https://goo.gl/4m9LNs>)
- How to Plan a Mobile App Project (<http://goo.gl/iILTx>)

### 3단계: 나의 앱을 코딩하자

앱 개발에 뛰어들 후에는 자신만의 앱을 개발하고 싶을 것이다. 그 난이도는 자신의 코딩 경험과 수준에 달렸지만 애플은 iOS와 스위프트로 그 장벽을 낮췄

다. 애플의 자체 IDE(Integrated Development Environment)인 엑스코드를 이용하면 맥에서 아이폰과 아이패드용 앱을 개발할 수 있다. 이밖에 스탠퍼드(Stanford), MIT, 하버드 등 대학이 제공하는 다양한 온라인 동영상 강좌도 있다. 아이튠즈 U(iTunes U)로 스탠퍼드의 iOS용 앱 개발 과정을 확인해 보자. 일반적인 개발에 대해 한 발 더 다가간 자신을 발견할 수 있을 것이다. 이 과정에서 참고할만한 자료는 다음과 같다.

- A Swift Tour (<https://goo.gl/kzvBRr>)
- Codecademy (<https://www.codecademy.com>)
- An Absolute Beginner's Guide to Swift (<http://goo.gl/a8Ao88>)
- The Swift Programming Language (<https://goo.gl/2Lf9yl>)
- Stanford's Developing Apps for iOS course in iTunes U (<https://goo.gl/wAVjx2>)




#### 4단계: 앱 개발자 고용

자신이 뛰어난 개발자가 아니라는 생각이 든다면 개발자를 고용(또는 동업)할 수 있다. 앱 개발을 위해 전문 개발자를 고용하면 비용이 많이 들 수 있다(일반적으로 약 2만 달러부터 시작된다). 하지만 충분히 흥미로운 프로젝트가 있다면 자신의 기술을 연마하고 싶은 개발자와 팀을 구성할 수 있다. 개발자들이 모이는 밋업(Meetup) 등의 사이트를 이용해 친목을 다지는 것도 좋은 방법이다. 프로젝트 예산이 이미 확보된 상태

라면 링크트인(LinkedIn) 등의 서비스를 사용해 개발자를 찾는 것이 더 유용하다.

#### 5단계: 아이튠즈 커넥트(iTunes Connect)로 앱 스토어에 제출

앱 개발이 끝났으면 이제 아이튠즈 커넥트를 통해 애플에 제출할 수 있다. ADM과는 별도로 아이튠즈 커넥트에 가입해야 한다. 애플은 앱 제출에 관한 포괄적인 지침이 있다. 애플이 한 가지 이유만으로도 앱 등록을 거절하는 경우가 자주 있으니 주의해야 한다. 허용되지 않은 SDK 기능을 사용하거나 알몸 노출, 포르노 등이 그 이유이다. 하지만 기술적인 부분도 어느 정도 주의해야 한다. 

# 애플 워치용 앱 개발 입문자를 위한 팁 모음

Nik Rawlinson | Macworld UK

2014년 WWDC의 가장 큰 화제 중 하나는 애플의 최 소형 제품인 '애플 워치(Apple Watch)'와 운영체제인 워치OS였다. 이후 업체는 꾸준히 워치OS를 개선했고 지난해 가을 '워치OS 2'를 발표했다. 워치OS 2는 이전 버전과 비교해서 어떤 차이가 있을까. 개발자가 이용할 수 있는 부분은 무엇일까. 워치OS 2의 가장 큰 특징은 개발자가 앱의 '지능'과 관련된 책임을 아이폰에서 워치 자체로 더 많이 옮기도록 허용한 것이다. 이를 통해 하드웨어의 핵심 기능을 더 많이 이용할 수 있게 됐다. 이는 워치를 블루투스를 통해 아이폰에 연동하는 종속적인 기기에서, 다양한 작업을 수행할 수 있는 손목 위의 완전한 컴퓨터로 바꿔 놓는다.

워치OS 2를 이용하면 개발자는 디지털 크라운(디지털 용두)과 마이크론 등 핵심 부품을 이용할 수 있다. 아이폰이 멀리 떨어져 있을 때도 메모를 저장했다가 이후 가까워지면 자동으로 연결해 메모를 옮기는 식의 음성 메모 기록 앱을 개발할 수 있다는 의미이다. 또한, 워치OS 2는 내부 오디오 시스템을 사용할 수 있도록 허용한다. 사용자가 기록한 메모를 즉시 재생하거나, 메모로 기록한 해야 할 일의 시간이 되면 이를 소리로 알려주는 것도 가능해졌다.

## 컴플리케이션

엑스코드에 도입된 클락키트(ClockKit)는 애플이 컴플리케이션(Complications)이라고 부르는 요소를 이용해 워치 페이스 자체를 제어할 수 있도록 한다. 이는 다이얼 옆에 위치해, 설치한 앱에서 또는 워치로 동기화된 정보를 보여주는 새로운 리드아웃(정보 제공 요소)이다. 앱에 컴플리케이션을 프로그래밍하면, 사용자가 앱을 열지 않고도 알림과 업데이트를 확인할 수 있다.

폴크스바겐의 카넷(Car-Net) 앱을 이용하면 원격으로 차량의 문을 잠그고, 주차장에서 차량을 찾을 수 있다. 또 전기 자동차인 e-골프(e-Golf)의 충전 상태를 모니터, VW 로고에 컴플리케이션으로 탑재된 동근 막대를 통해 배터리 잔량을 볼 수 있다. 디지털 크라운을 위 또는 아래로 돌리면, 막대가 현재의 충전 상태 대신 선택한 시간에 예상되는 배터리 잔량을 표시한다. 이를 통해 목적지에 갈 수 있는 충분한 배터리가 있는지 미리 확인할 수 있다.

참고로 애플이 이런 추가 정보 표시에 '컴플리케이션'이라는 위압적인 이름을

붙인 이유가 있다. 사실 이는 시계 업계에서 수 세기 동안 사용한 용어이다. 날 짜나 달력 등 시간 외의 추가 정보를 뜻하는 말이다.

### 워치OS 프로그램 개발의 기회와 장점

워치OS 2를 통해 추가적인 가능성이 열린 것은 개발자에게 큰 희소식이다. 마치 애플이 아이폰 3을 출시하면서 웹용 앱을 탈피해 아이폰용 SDK(Software Development Kit)를 출시하기로 한 것과 비슷하다. 이는 워치를 더욱 매력적인 플랫폼을 만든다. 특히 앱 시장에서의 경쟁이 아직 iOS나 OS X만큼 치열하지 않다는 것이 장점이다.

그렇다면 워치용 앱 개발은 어떻게 시작해야 할까. 엑스코드는 워치OS, iOS, OS X용 소프트웨어를 개발할 때 사용하는 개발 도구이다. 맥 앱 스토어에서 최신 버전을 무료로 내려받을 수 있다. 애플 개발자 프로그램(Apple Developer Program)에 가입하면 다음 버전의 베타판도 얻을 수 있다. 엑스코드 6 버전부터 스위프트를 지원한다. 스위프트 2는 엑스코드 7 베타 버전부터 지원하니 참고하기 바란다. 엑스코드는 개발자가 시각적인 배치를 디자인하고, 이를 구현하는 코드를 작성할 수 있도록 도와준다. 오류를 점검하고, 결과물을 스스로 이용하거나 앱 스토어 또는 맥 앱 스토어를 통해 판매할 수 있는 실행 런타임으로 컴파일하는 것도 엑스코드의 역할이다.



### 스위프트를 배우는 방법

최상의 스위프트 입문서는 애플이 직접 만든 문서(<https://goo.gl/DmmCcQ>)다. C 관련 언어를 알고 있다면 이해하는 데 도움이 되지만 컴퓨터 언어에 대한 사전 지식이 없어도 무방하다. 단계별로 상수와 변수 등 스위프트 언어의 중요 특징을 소개한다. 단계별 학습이므로 필요할 때 중단했다가 다시 시작할 수 있다. 본업에 충실하면서 자투리 시간을 이용해 새로운 기술을 배울 수 있도록 쉽게 구성됐다. 혹시 브라우저 창과 코딩 공부를 할 수 있는 환경을 오가기가 불편하다면 아이북스(iBooks)에서 애플이 무료로 배포하는 521페이지 분량의 'The Swift Programming Language'를 내려받는 것(<https://goo.gl/0BGfRl>)도 좋다. 스위프트 2 관련된 최신 내용이 포함돼 있다. 새로운 기술을 터득하는 최고의 방법의 하나는 온라인 강좌다. 유데미에 올라온 Complete Apple Watch Developer Course(<https://goo.gl/ko1Ns2>) 강좌는 현재 구독자가 1만 명이 넘는다. 18시간 143개 강의로 구성돼 있으며 개발 경험이 없어도 즉시 학습을 할 수 있다.

기존에 아이폰이나 맥 프로그래밍 경험이 없는 사람이 애플 워치 앱 개발을 시작한다면 첫 앱은 느리면서도 꾸준하게 개발하는 것이 좋다. 초기에는 단계별로 작게 시작하는 것이 나중에 배운 것을 종합할 때 도움이 될 것이다. 또 판매가 아닌 개인적으로 사용할 프로젝트를 출발점으로 삼는 것이 낙담하지 않으면서 기술을 연마하는 방법이다. ITWORLD

# 윈도우 개발자도 주목해야 할 스위프트의 변화

Simon Bisson | InfoWorld

지난 12월 애플은 약속한 대로 스위프트(Swift) 언어를 오픈소스화했다. 2014년에 첫선을 보인 스위프트는 범용 애플리케이션을 개발할 수 있는 C 계열 언어다. 애플은 새로운 데스크톱, 모바일 앱 개발에 초점을 두고 LLVM 컴파일러를 사용하는 언어로 스위프트를 만들었다. 그 결과 C의 정교함과 깊이는 물론 파이썬과 같은 인터프리트 언어의 쉬운 사용성을 결합하면서, 동시에 현대 언어 설계 사조를 폭넓게 적용한 새로운 언어가 탄생했다.

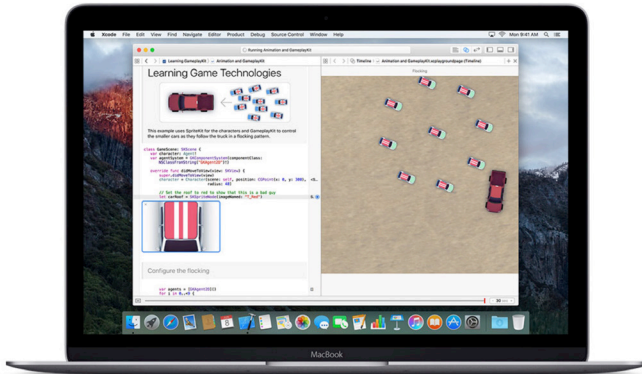
이번에 스위프트가 오픈소스로 공개됨에 따라 리눅스 진영의 지원과 서버 측 언어 기능이 추가되는 등 지속해서 개발이 이루어질 것으로 기대된다. 스위프트는 이미 빠른 속도로 발전하고 있다. 나온 지 1년여 만에 버전 2.2에 이르렀고 LLVM과 클랭(clang) 컴파일러에 전용 기능이 추가됐다. 아직 윈도우 버전은 없지만, 마이크로소프트는 비주얼 스튜디오 최근 버전에 iOS와 OS X 애플리케이션에 대한 스위프트 지원을 추가했다.

## 기존 애플 개발자가 스위프트에 관심을 가져야 하는 이유

OS X에서 스위프트를 사용한다면 개발 환경은 기존과 거의 차이가 없다. 오브젝티브-C 개발자에게 오래전부터 익숙한 개발 환경이며, 두 언어의 컴파일러도 같다. 애플은 이를 한마디로 잘 정리했었다. “스위프트는 C 없는 오브젝티브-C”라고. 그로 그럴 것이 스위프트는 오브젝티브-C와 기반 LLVM 컴파일러의 추출물이라고 할 수 있다. 오브젝티브-C에서 특정 키 구조를 간소화하고 코드의 가독성을 더 높인 것이 곧 스위프트라고 할 수 있을 정도다.

스위프트는 오브젝티브-C의 구문을 기반으로 하며, 자바스크립트와 C# 진영에서 건너오는 iOS 또는 OS X 개발자 관점에서 접근성이 좋고 단순한 현대적인 구조를 지원한다. 개발자는 스위프트에서 제어 흐름을 처리하는 방식부터 함수 사용법에 이르기까지 익숙한 요소를 많이 발견할 것이다. 스위프트와 오브젝티브-C의 유사성 덕분에 스위프트로 앱을 업데이트하는 과정도 간편하다. 같은 앱에서 스위프트와 오브젝티브-C 코드를 함께 사용할 수 있으므로 새 앱에서 기존 라이브러리를 재활용할 수 있다. 둘 중 한 언어에서 다른 한 언어로 바꿀 때 재사용할 코드를 굳이 다시 쓸 필요가 없다.

애플의 설계 원칙은 안전성에 초점을 두고 있으며 스위프트의 주요 특징 중



하나는 형식 상속 지원이다. 그 결과 스위프트는 강력한 형식 언어가 아니지만, 형식에 안전한 언어다. 변수 형식을 선언하지 않았더라도 이미 선언된 값으로 이 변수를 설정하는 경우 스위프트는 이를 선언된 형식으로 추론한다. 따라서 코드의 안전성이 높아지고 충돌 위험이 낮아진다.

형식 안전성은 곧 int를 string으로 전달할 수 없음을 의미한다. 스위프트 컴파일러는 이러한

경우를 항상 오류로 잡아낸다. guard 문은 값이 nil인 경우 해당 코드 섹션 실행을 중지시키므로 개발자는 일반적인 사용자와 API 오류(보통 nil은 컴파일러 타임 오류를 일으킴)로부터 코드를 보호할 수 있다. 또는 optionals를 사용해서 어떤 형식을 구현하든 관계없이 특정 변수에 null을 허용할 수 있다.

### 비애플 개발자가 관심을 가질 기능들

스위프트로 앱을 만들기는 아주 쉽다. 바로 시작하면 된다. main 함수를 준비하거나 참조를 가져올 필요가 없다. 이것은 코딩의 접근성을 높이도록 설계했기 때문이다. OS X 또는 iOS 사용자 인터페이스의 대부분은 라이브러리로 처리되므로, 모양을 입히고 꾸미기 전에 앱의 핵심부터 바로 시작할 수 있다.

스위프트의 핵심 요소 중 하나는 애플의 표현을 그대로 옮기자면 '프로토콜 지향 프로그래밍'이다. 다른 언어의 인터페이스와 긴밀하게 연관되는 프로토콜에는 특정 메소드를 구현하기 위한 클래스가 필요하다. 따라서 프로토콜을 사용하는 클래스에서 언제든지 정의된 메소드를 사용할 수 있다. 소스 코드를 편집할 필요 없이 프로토콜을 통해 핵심 언어 요소를 포함한 모든 클래스를 확장할 수 있다. 이는 서드파티 코드와 라이브러리로 작업할 수 있음을 의미한다. 또 다른 큰 장점은 개발팀에게 코딩 표준을 구현하는 프로토콜 집합을 제공할 수 있다는 점이다. 프로토콜은 아주 강력한 도구로, 제네릭(Generic)을 개발 전략의 중요한 요소로 만들어준다.

스위프트에서 가장 중요한 혁신은 엑스코드에 플레이그라운드(Playground)를 추가한 것이다. 애플은 엑스코드와 스위프트를 사용해 LLVM 컴파일러의 더 많은 부분을 개발자에게 공개하면서 IDE 내에서 코드가 하는 일을 볼 수 있게 했다. 개발자는 IDE에서 나올 필요 없이 플레이그라운드를 통해 코드의 특정 섹션을 신속하게 실행해서 원하는 대로 작동하는지를 확인할 수 있다. 코드를 컴파일하고 빌드 또는 디버거를 실행할 필요가 없다. 작성한 코드 옆에 모두 제공하기 때문이다. 이러한 구조의 목적은 프로그래밍, 말 그대로 코드 놀이(play)를 촉진하는 데 있다.



### 애플 개발 플랫폼의 확장 가능성

이와 같은 기능을 다른 플랫폼으로 가져오는 것도 중요한 일이다. 언어는 클라이언트에서 서버, 클라우드에 이르기까지 개발자가 어디에서 일하든 사용 가능해야 하며, 효율적인 개발을 위해서는 현재 사용되는 스택의 모든 요소에서 일관성을 확보하는 것이 핵심이다. 리눅스에서는 스위프트 컴파일러에 플레이그라운드도 없다.

그러나 이쪽에서는 서버 애플리케이션 개발이 주를 이루므로 사실 출력을 콘솔로 넘기는 것 이상으로 신속하게 함수를 시각화할 필요가 없다. 애플은 상호작용하는 방식으로 신속하게 코드를 다룰 수 있도록 리눅스 도구에 REPL 기반 디버거를 제공한다.

이제 스위프트 소스 코드가 깃 허브(GitHub)에 공개되어 있으므로 누구나 원본 리포지토리를 가져와 코드를 실험하고 스위프트에 코드 기여를 요청할 수 있다. 아마도 여기서 가장 중요한 요소는 애플 개발 프레임워크를 다른 플랫폼으로 확장할 가능성을 열어주는 스위프트의 코어 라이브러리일 것이다. 스위프트 코드를 한 번만 작성하면 여러 플랫폼에 제공할 수 있는 발판이 마련됐다. 이렇게 되면 개발자는 마이크로소프트 오픈소스 .Net 외의 대안을 갖게 된다. 자마린(Xamarin)과 같은 크로스 플랫폼 도구 기업은 훨씬 더 폭넓은 개발자 집단을 지원할 수 있게 된다. 