공개 SW 개발자대회

THIS IS THE PY
https://github.com/thisisthepy

PyComposeUI 개발팀

온디바이스 인공지능 개발 및 테스트용 REPL 환경 개발

Organization Introduction

thisisthepy 프로젝트

온디바이스 인공지능 개발을 위한
파이썬 멀티플랫폼 개발 솔루션의 집합



thisisthepy

The efforts to use Python everywhere

9 followers Korea, South thisisthepy@gmail.com



9 followers · Korea, South · thisisthepy@gmail.com

Pinned

python-multiplatform-mobile — Public

A multiplatform solution to use Python with Kotlin interoperably.

C · ☆ 6

pmm-app-template — Public template

Python Multiplatform Mobile (PMM) Application Project Template with Kotlin Multiplatform Mobile

Kotlin · ☆ 7 · 3

pmm-toolchain — Public

Python Multiplatform Build Tool with Kotlin Multiplatform Mobile

Python · ☆ 5

pythonx-pycomposeui — Public

Python Wrapper for Jetpack Compose Multiplatform Mobile

Python · ☆ 6

pycomposeui-hot-reloader — Public

Hot Reload (Live Update) Server & Client for PyComposeUI Application

Python · ☆ 4

PyREPL — Public

UI-Enabled Python REPL for Android/iOS/Desktop

CSS · ☆ 5 · 8

Table of contents

1 개발 배경 및 목적
Background/Motivation

2 Python Multiplatform & Kotlin-Python 바인더

3 멀티플랫폼 개발 보조 도구 (PythonX, PyREPL)

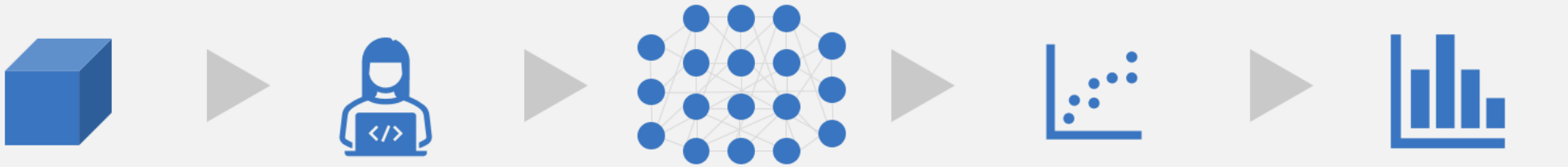
4 기대효과 및 활용 분야
Usage/Applications

Part 1

개발 배경 및 목적

Background/Motivation

현재의 온디바이스 인공지능 개발 과정



개발 과정과 배포과정의 불일치로, 사용하는 언어가 달라지고,
일부 코드를 재작성하기도 해야 하면서 개발 경험이 일관적이지 못한 상황

발생 가능한 문제점

– 개발 언어와, 런타임에서 사용하는 언어가 달라 디버깅 과정에서의 어려움 발생

Part 1
개발 배경 및 목적

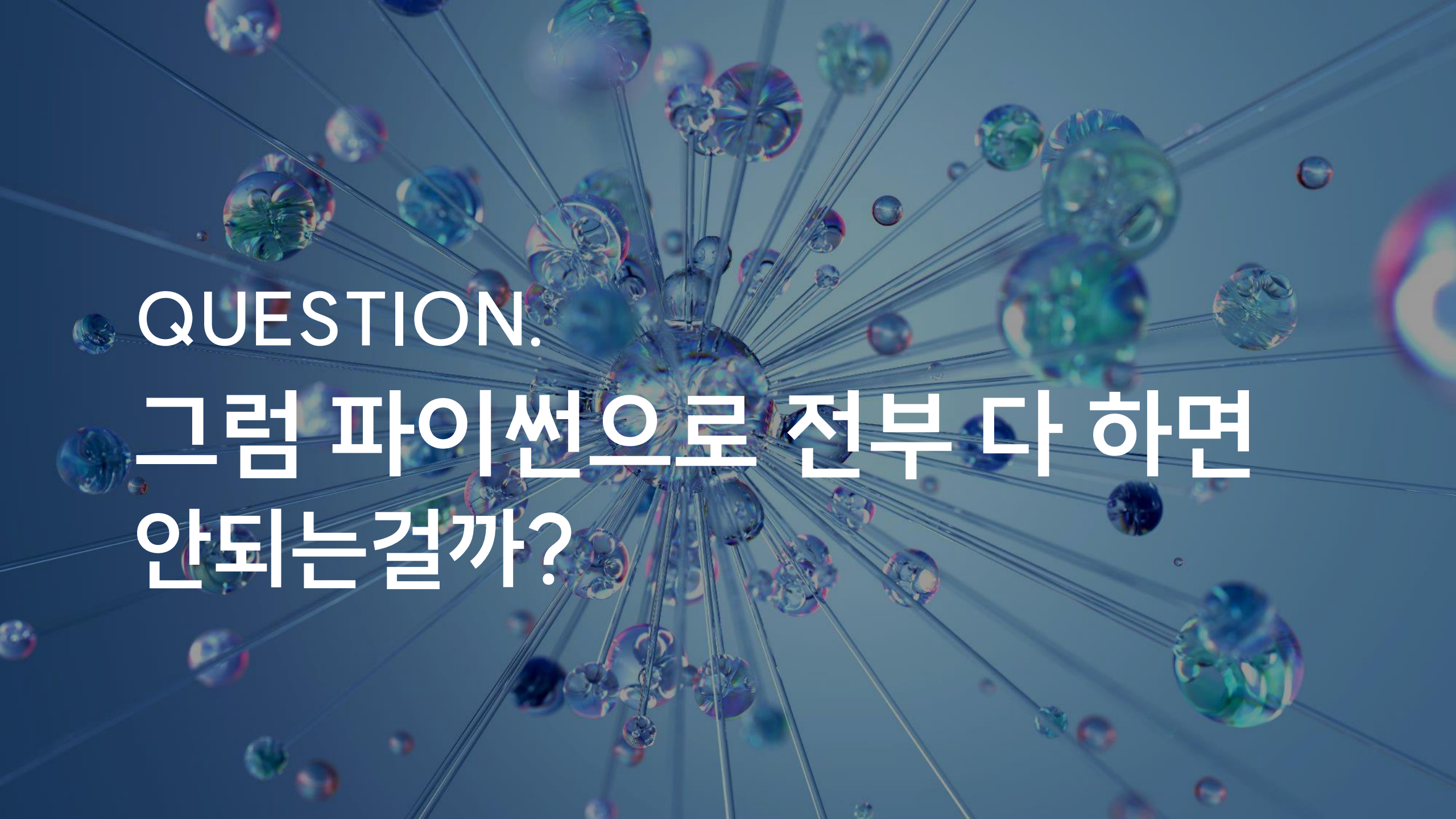
현재의 온디바이스 인공지능 개발 과정



개발 과정과 배포과정의 불일치로, 사용하는 언어가 달라지고,
일부 코드를 재작성하기도 해야 하면서 개발 경험이 일관적이지 못한 상황

– 언어에 상관없이 호환되도록 하는 목표를 가진 공통 포맷이 존재하기는 하나, 모든 레이어가 호환되지 않고, 특히 새로운 레이어 구조를 적용한 경우, 변환 자체에 어려움이 존재

=> 변환 툴이 어떻게 변환을 시도하는지 까지도 전부 알고 있어야 정상적으로 모델을 변환할 수 있기 때문에 난이도가 높으며, 변환이 필요하다는 것 만으로도 개발 난이도를 증가시키는 원인으로 작동


QUESTION.
그럼 파이썬으로 전부 다 하면
안되는걸까?

*왜 파이썬은 모바일 개발용으로 잘 쓰이지 않을까요?

*파이썬은 안 쓰이는 곳 없이 매우 다양한 분야에서 활용되고 있는 언어지만, UI 개발 분야에서는 그 저력을 보이지 못하고 있습니다.

지금까지의 파이썬 생태계의 문제점

1. 단일화 되지 않은 멀티플랫폼용 파이썬 임베딩 방식

   - 공식적인 Android, iOS 용 파이썬 빌드 방식이 그동안 부재하였음.
   - Python 3.13부터 Android/iOS가 공식 지원 플랫폼으로 추가되었으나, 파이썬 단일로는 모바일 운영체제의 API를 호출하기 힘들고, 멀티플랫폼용 바인더는 존재하지 않았음.

2. 현대적인 UI 개발 프레임워크의 부재

   - 현대적인 선언형 UI 개발 프레임워크는 존재하지 않고, 모두 이전에 만들어진 Legacy만 존재하여 실사용이 어려움.

*왜 파이썬은 모바일 개발용으로 잘 쓰이지 않을까요?

*파이썬은 안 쓰이는 곳 없이 매우 다양한 분야에서 활용되고 있는 언어지만, UI 개발 분야에서는 그 저력을 보이지 못하고 있습니다.

지금까지의 파이썬 생태계의 문제점

3. 불편한 End-User용 배포 패키징 솔루션

   – 파이썬은 태생적으로 하나의 단일 파일로 배포하는 것이 힘든데, 이를 가능하게 할 편리한 서드 파티 멀티플랫폼 패키징 툴도 존재하지 않음.

*왜 파이썬은 모바일 개발용으로 잘 쓰이지 않을까요?

*파이썬은 안 쓰이는 곳 없이 매우 다양한 분야에서 활용되고 있는 언어지만, UI 개발 분야에서는 그 저력을 보이지 못하고 있습니다.

발견된 니즈

- 이와 같은 상황으로 인해 파이썬을 주로 사용하는 인공지능이나 데이터 사이언스 분야의 사람들은 직접 만든 파이썬 코드를 이용하여 End-User에게 바로 배포하지 못하고 다른 언어를 배워서 추가적인 작업을 더 진행해야 하는 형국.

- 좋은 AI 솔루션을 개발한다고 해도, 실제 사용자가 사용할 수 있도록 하기 위해서는 UI등의 시스템이 필요하기 때문에 파이썬을 통한 UI 개발에 새로운 방안을 제시한다면 많은 사람들이 혜택을 입을 수 있을 것으로 보았고, 필요성을 인지하여 프로젝트를 기획하게 됨.

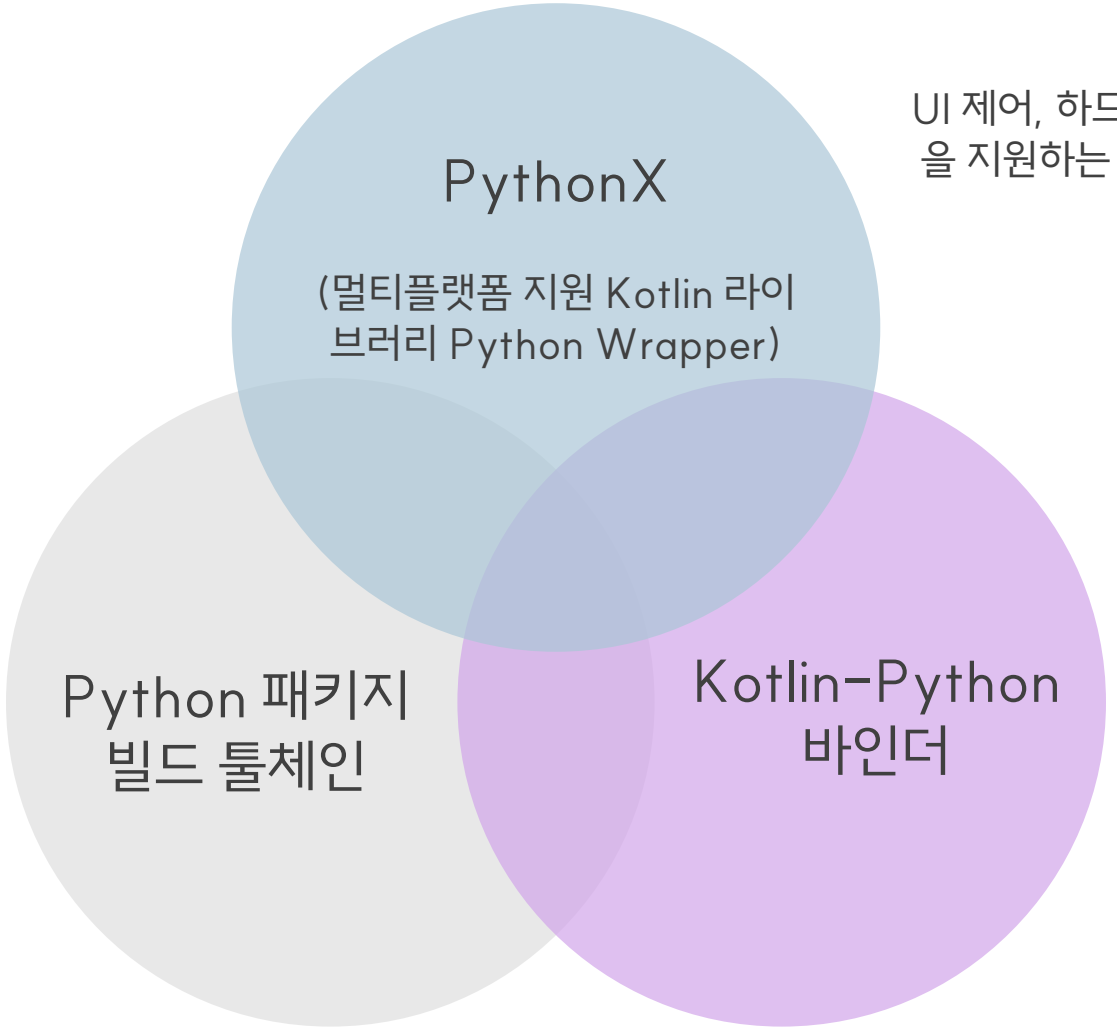
솔루션 제시 – PythonMultiplatform

1. 저희는 이전의 솔루션들이 겪은 문제점을 해소하고 더 나은 개발 방식을 제공하기 위해, 파이썬을 단일로 사용하기 보다는 다중 플랫폼 환경에서 사용 가능한 Kotlin Multiplatform Mobile과 바인딩을 진행함으로써 코틀린과 파이썬이 상호 작동 할 수 있도록 하여, 서로의 라이브러리들을 공유하며 사용할 수 있도록 하는 방식을 택하여 확장성을 높였고,

2. 최신 기술인 Compose Multiplatform을 파이썬 라이브러리로 Wrapping하여 현대적인 선언형 UI를 최초로 파이썬을 통해 사용할 수 있도록 하기로 기획하였습니다.

3. 또한, 파이썬 생태계에서 매우 큰 장점으로 꼽히는 환경인 Jupyter Lab을 통한 모바일 개발 또한 지원하여, 파이썬을 사용하던 사람들이 모바일 환경에서의 개발을 처음 접함에 있어 이질감을 줄이기 위해 노력하고 있습니다.

[프로젝트 구성요소]



UI 제어, 하드웨어 제어(센서, 카메라, 진동 등) 등을 지원하는 타겟 플랫폼별 Kotlin 라이브러리의 Python Wrapping

Gradle과 통합된 Python 패키지 빌드 툴 체인 및 실행 파일 생성기, 의존성 관리자

Kotlin-Python 간 클래스 및 변수 동적 바인더, 정적 타입 추론 메타 패키지 제너레이터


Part 2
Python Multiplatform &
Kotlin-Python 바인더

QUESTION.

왜 파이썬으로는 모바일 개발을 잘 하지 않을까?

이전 솔루션 분석 – Kivy

- 파이썬 개발자 위주의 경험 (pip로 빌드 도구 설치 후 이용하는 형태)
- 파이썬 코드를 작성한 후 터미널에서 빌드 도구를 이용하여 플랫폼별 실행 파일을 생성하는데, 이 때 실제 플랫폼별 빌드 툴이 내부적으로만 사용되도록 숨겨져 있어 세부적인 설정을 수정하는 데 어려움 발생

=> 결론적으로 초보자들을 위해 많은 부분을 추상화 하였으나, 그 때문에 발생하는 직접적인 플랫폼별 코드/빌드 시스템에 접근이 어려워져 기업 등의 프로 사용자들이 도입하기에는 어려운 솔루션이 되었음

=> 또한, 파이썬'만'으로 앱을 개발하는 것이 목표였기에 각 운영체제의 네이티브 코드와는 통합이 어려운 형태라 네이티브 솔루션과 함께 결합하는 형태로 사용이 불가능했음.

이전 솔루션 분석 – BeeWare

- 파이썬 개발자 위주의 경험 (Kivy와 동일한 빌드 도구상의 문제)
- 파편화된 파이썬 임베딩 (각 플랫폼별로 완전히 상이한 코드 베이스로 빌드됨)

Python 3.13 출시

- Android와 iOS가 Python의 Teer 3 공식 지원 플랫폼으로 승격됨 (단, 빌드 완료된 바이너리는 공식 배포하지 않음)
- JIT 컴파일 지원 시작
- GIL 선택적 활성화 혹은 Sub-Interpreter 사용 가능

=> 기존에 문제점으로 제시되던 파이썬의 성능 관련 문제가 해결되기 시작하는 시점

=> 이제 파이썬 자체적으로 모바일 앱 개발을 지원할 준비를 마쳤으나, 필연적으로 파이썬을 임베딩할 수 있는 서드파티 솔루션이 필요하게 됨

Python Multiplatform Mobile

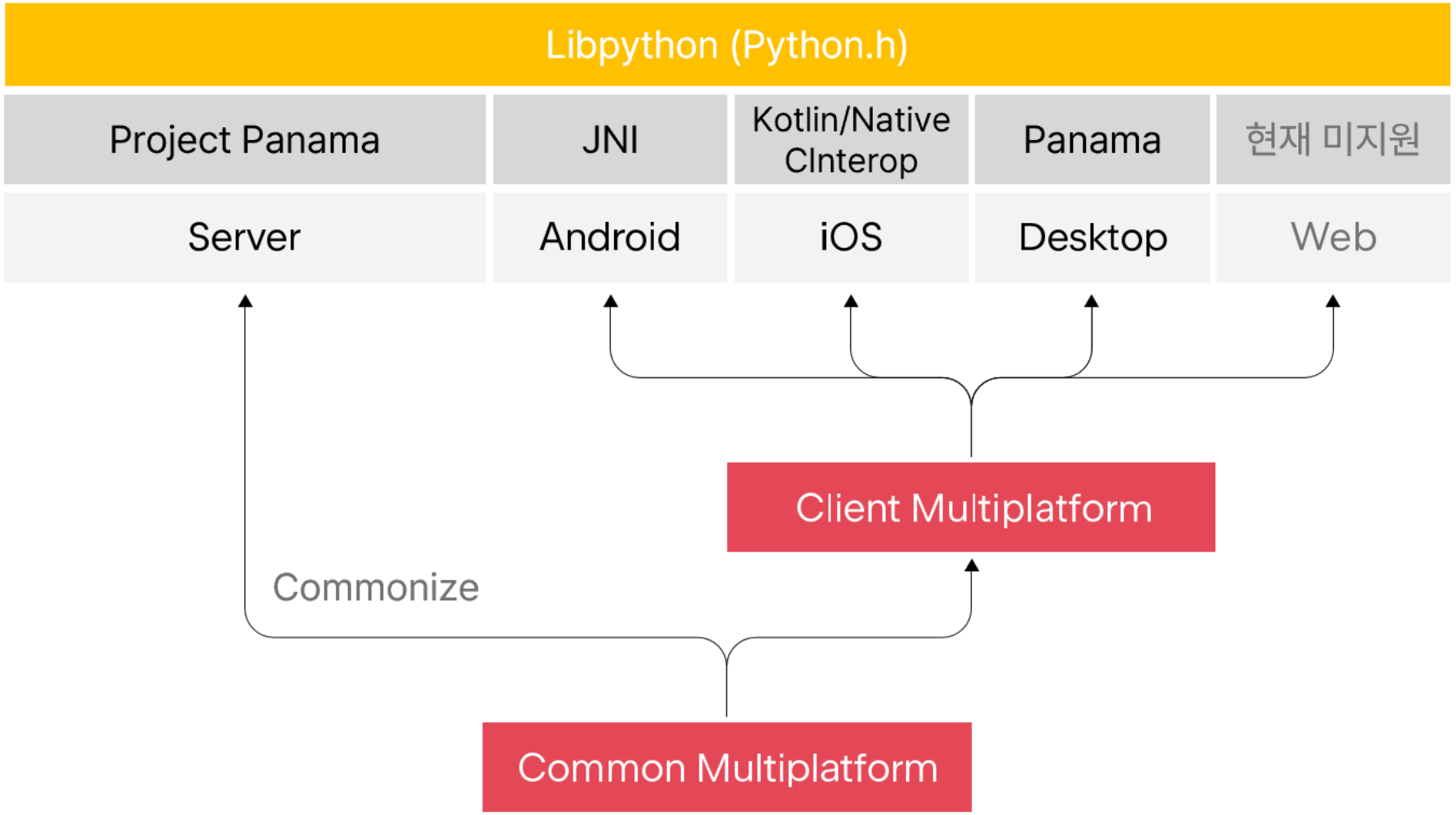
멀티 플랫폼을 위한 가장 엘레강스한 파이썬 임베딩

코틀린 프로그램에 파이썬 인터프리터를 내장하는 형태를 선택하여 네이티브 코드와 유현하게 통합 가능
(기존 솔루션들과는 반대 방향)

모바일 개발자들이 사용하는 Gradle을 빌드 도구로 선택함으로써, 모바일 개발자들이 이질감 없이 바로 솔루션을 원래 진행하던 프로젝트에 통합 가능 (기존 솔루션들을 프로 사용자들이 도입을 꺼리던 원인 해결)

상호 동적 바인딩을 지원하여 코틀린 변수/함수를 파이썬에서 접근하거나 그 반대를 모든 플랫폼에 대해서 지원 (기존 솔루션들은 플랫폼별로 통합된 형태로 바인딩을 제공하지 못하였음)

CPython 인터페이스 추상화 계층

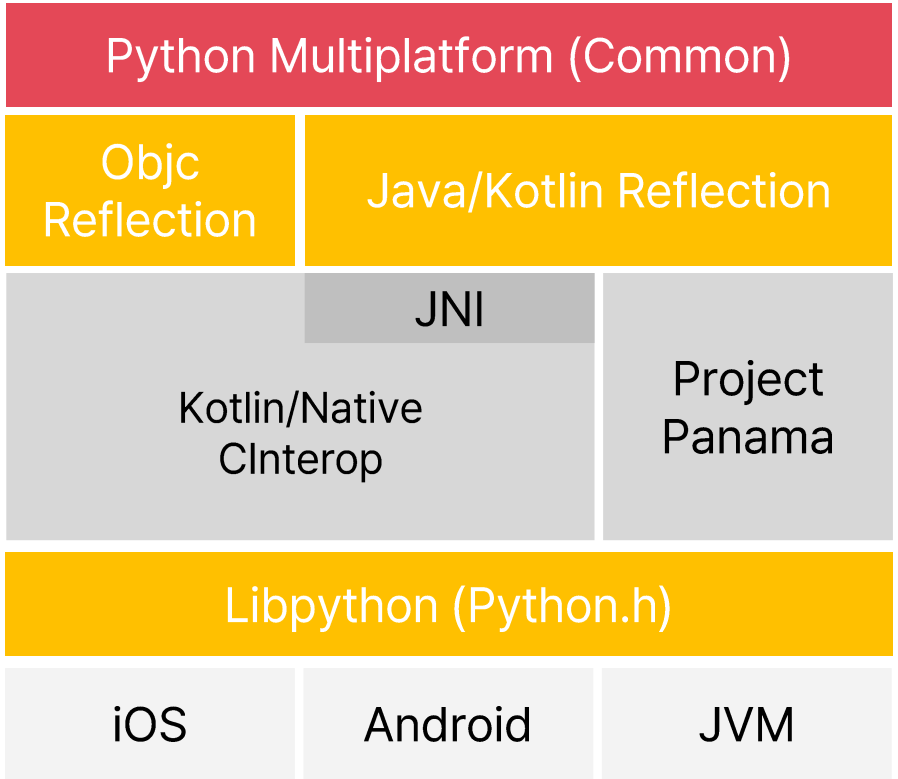


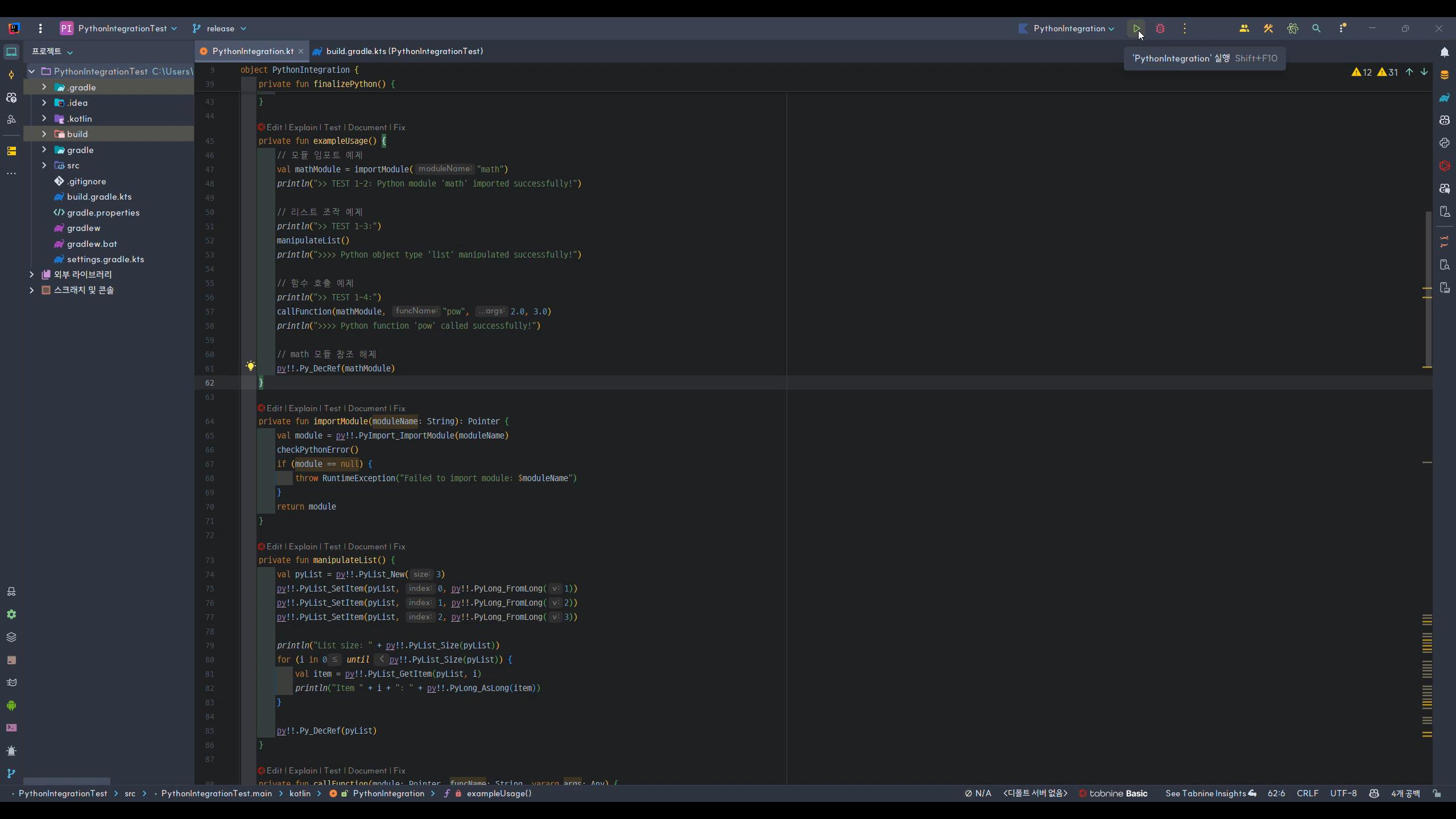
- 각 플랫폼별 CFFI를 통해 Cpython Embed API를 호출하는 함수들을 작성하고

- 이를 Kotlin Common Module로 추상화하여 동일 코드베이스인 것 처럼 동작하도록 설계

- 클래스 동적 검색을 위해 각 플랫폼 별 Reflection Method 사용

Binding 세부 구현도




PythonIntegrationTest
release
PythonIntegration
'PythonIntegration' 실행 Shift+F10
프로젝트
PythonIntegrationTest C:\Users\
.gradle
.idea
.kotlin
build
gradle
src
.gitignore
build.gradle.kts
gradle.properties
gradlew
gradlew.bat
settings.gradle.kts
외부 라이브러리
스크래치 및 콘솔
PythonIntegration.kt
build.gradle.kts (PythonIntegrationTest)
12 31
object PythonIntegration {
private fun finalizePython() {
}
Edit | Explain | Test | Document | Fix
private fun exampleUsage() {
// 모듈 임포트 예제
val mathModule = importModule(moduleName: "math")
println(">> TEST 1-2: Python module 'math' imported successfully!")
// 리스트 조작 예제
println(">> TEST 1-3:")
manipulateList()
println(">>>> Python object type 'list' manipulated successfully!")
// 함수 호출 예제
println(">> TEST 1-4:")
callFunction(mathModule, funcName: "pow", ...args: 2.0, 3.0)
println(">>>> Python function 'pow' called successfully!")
// math 모듈 참조 해제
py!!.Py_DecRef(mathModule)
}
Edit | Explain | Test | Document | Fix
private fun importModule(moduleName: String): Pointer {
val module = py!!.PyImport_ImportModule(moduleName)
checkPythonError()
if (module == null) {
throw RuntimeException("Failed to import module: $moduleName")
}
return module
}
Edit | Explain | Test | Document | Fix
private fun manipulateList() {
val pyList = py!!.PyList_New(size: 3)
py!!.PyList_SetItem(pyList, index: 0, py!!.PyLong_FromLong(v: 1))
py!!.PyList_SetItem(pyList, index: 1, py!!.PyLong_FromLong(v: 2))
py!!.PyList_SetItem(pyList, index: 2, py!!.PyLong_FromLong(v: 3))
println("List size: " + py!!.PyList_Size(pyList))
for (i in 0 ≤ until < py!!.PyList_Size(pyList)) {
val item = py!!.PyList_GetItem(pyList, i)
println("Item " + i + ": " + py!!.PyLong_AsLong(item))
}
py!!.Py_DecRef(pyList)
}
Edit | Explain | Test | Document | Fix
PythonIntegrationTest › src › PythonIntegrationTest.main › kotlin › PythonIntegration › exampleUsage()
Ø N/A
<디폴트 서버 없음>
tabnine Basic
See Tabnine Insights
62:6
CRLF
UTF-8
4개 공백



Part 3

멀티플랫폼 개발 보조 도구
(PythonX, PyREPL)


QUESTION.
앱 개발은 데스크탑에서
해야만 하는걸까?
폰에서 그냥 바로 하면 안되는건가?



QUESTION.

앱 개발은 왜, 코드 수정할 때 마다 빌드를 새로 해야 해?

PythonX

코틀린 라이브러리의 파이썬 어댑터

최소한도의 필수적인 Wrapping 제공
– 파이썬과 코틀린간의 언어 표현력의 차이를 해결하기 위한 어댑터 역할

UI 제어, 하드웨어 제어(센서, 카메라, 진동 등) 등을 지원하는 타겟 플랫폼별
Kotlin 라이브러리의 Python Wrapping을 통한 유틸성 코드 제공

PythonX-Compose

Kotlin Jetpack Compose를 파이썬으로 Wrapping

이전 솔루션 분석 – Kivy

- 자체적인 UI 제작 문법을 사용
- 멀티플랫폼 지원
- **문제점**

– 직접적으로 플랫폼별 코드에 접근하기 힘들어 Kivy에서 제공하는 추상화된 API외에는 운영체제별로 필요한 코드에 접근이 힘들었음 (Ex: 앱 권한 관리 등)

– 자체적인 UI 제작용 따로 배워야 하기 때문에 파이썬으로 온전히 코드를 작성하는 형태가 아닌데다, 결론적으로 선언형 UI 프레임워크가 아님

이전 솔루션 분석 – Beeware Toga

- 선언형 UI를 지원하지 않음

이전 솔루션 분석 – Flet

- 파이썬을 위한 Flutter Wrapper
- Dart 문법을 모르더라도 파이썬 만으로도 Flutter 앱을 작성할 수 있도록 지원
- **문제점**
  - 랩핑 과정에서 선언형 UI 특징이 사라지고 명령형 형태로 바뀜
  - 파이썬과 Flutter앱이 사실은 하나의 앱이 아니라 서로 다른 프로세스가 소켓 통신을 하는 형태로 구현되어 비효율적 (실제 사용되는 앱 개발용으로 채택하기 어려움)
  - 실제로는 멀티플랫폼이 아닌, WebView 형태로 제공되어 사용자 경험을 해치게 됨

Part 3

멀티플랫폼 개발 보조 도구 (PythonX, PyREPL)

```
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.bodyLarge,
                )
            }
        }
    }
}
```



X

Jetpack Compose는 코틀린으로 작성된 최신 선언형 UI 멀티플랫폼 프레임워크로, 하나의 UI 로드와 비즈니스 로직으로 멀티플랫폼 앱을 만들 수 있도록 지원하는 것이 가장 큰 특징

선언형 패러다임을 채택하여, 매우 현대적이고 재사용성 높은 UI 코드를 작성할 수 있도록 지원하고 있음

X

PythonX 라이브러리는 이러한 코틀린의 라이브러리와 프레임워크를 파이썬으로 Wrapping하여 코틀린의 개발 경험을 파이썬으로 가져오는 역할을 수행하도록 설계되었음.

Part 3

멀티플랫폼 개발 보조 도구 (PythonX, PyREPL)

```kotlin
@Composable
fun JetpackCompose() {
    Card {
        var expanded by remember { mutableStateOf(false) }
        Column(Modifier.clickable { expanded = !expanded }) {
            Image(painterResource(R.drawable.jetpack_compose))
            AnimatedVisibility(expanded) {
                Text(
                    text = "Jetpack Compose",
                    style = MaterialTheme.typography.bodyLarge,
                )
            }
        }
    }
}
```

Kotlin

```python
@Composable 신규*
def MainView():
    Card(
        content=lambda: {
            Column(
                modifier=Modifier.fillMaxSize(),
                content=lambda: {
                    Text("PyComposeUI PyREPL 데모에 오신 것을 환영합니다!")
                }
            )
        }
    )
```

Python

X

코틀린 사이드에서 UI를 위해 작성하는 Composable 함수 생성 방식과 유사하게, 파이썬에서도 @Composable 데코레이터를 제공하여, 단순히 함수에 데코레이터를 붙이는 것 만으로도 UI를 그리는 함수로 변환되도록 설계하였음.

PyREPL

파이썬 사용자들이 애용하는 Jupyter Lab을 모바일 운영체제에서도 동작할 수 있게끔 포팅하여 데스트탑에서의 파이썬 개발 경험을 모바일용 어플리케이션 개발에도 유지할 수 있도록 지원

Jupyter Lab에서도 모바일 어플리케이션 UI를 제어할 수 있도록 기능을 추가
(UI 스레드에서 주피터 커널을 실행)

어플리케이션 개발을 모바일 기기 만으로도 가능하게 지원

주피터에서 직접 UI Composable을 작성하고 화면에 즉시 반영할 수 있게 하여 UI 코드의 변경 사항을 즉시 확인 가능

Part 3

멀티플랫폼 개발 보조 도구 (PythonX, PyREPL)

A

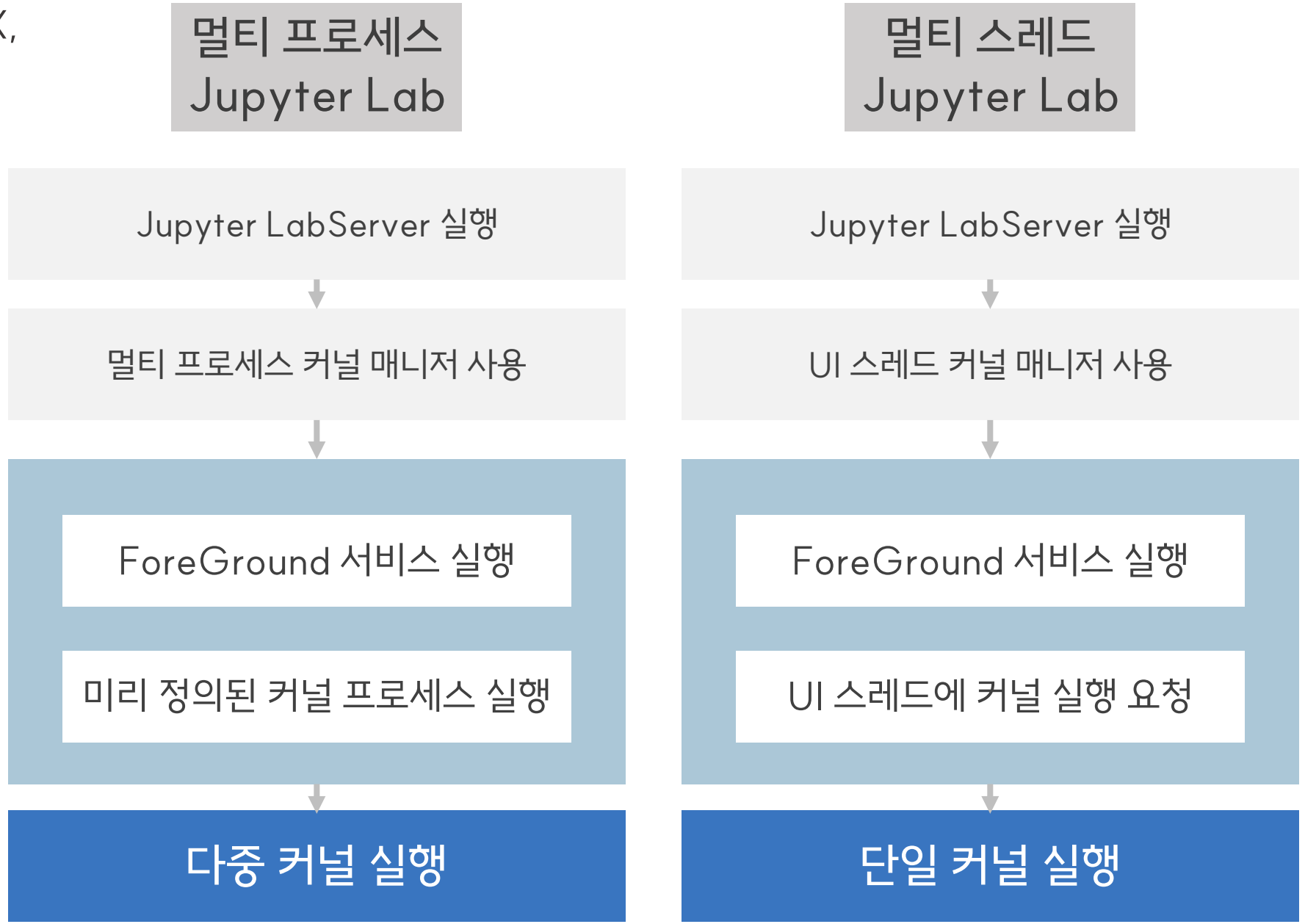
멀티 프로세스
Jupyter Lab

다중 커널 사용 가능

B

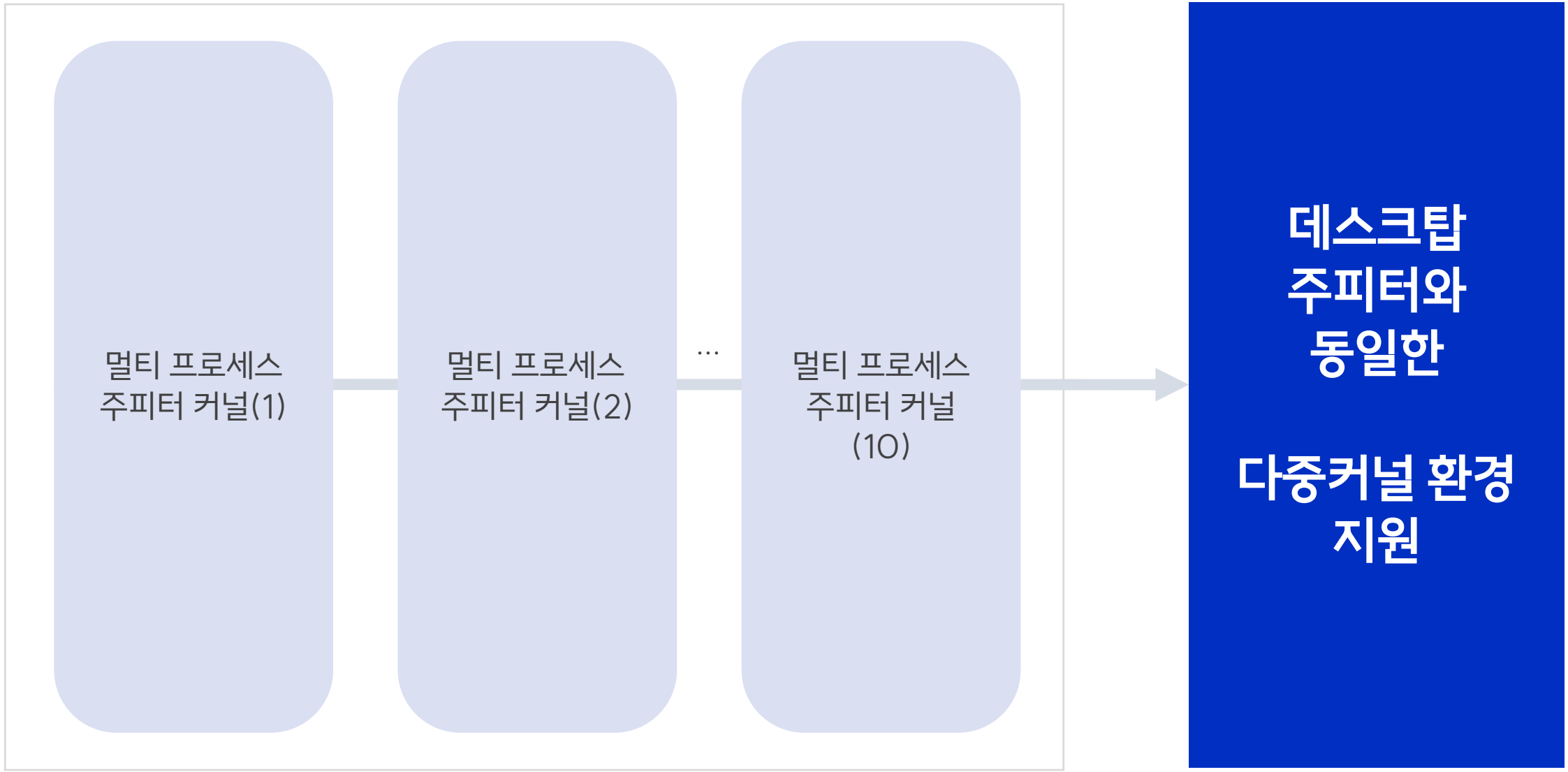
멀티 스레드
Jupyter Lab

단일 커널로 사용 제한
But, UI 스레드 변수 접근 가능


멀티 프로세스
Jupyter Lab
Jupyter LabServer 실행
멀티 프로세스 커널 매니저 사용
ForeGround 서비스 실행
미리 정의된 커널 프로세스 실행
다중 커널 실행
멀티 스레드
Jupyter Lab
Jupyter LabServer 실행
UI 스레드 커널 매니저 사용
ForeGround 서비스 실행
UI 스레드에 커널 실행 요청
단일 커널 실행

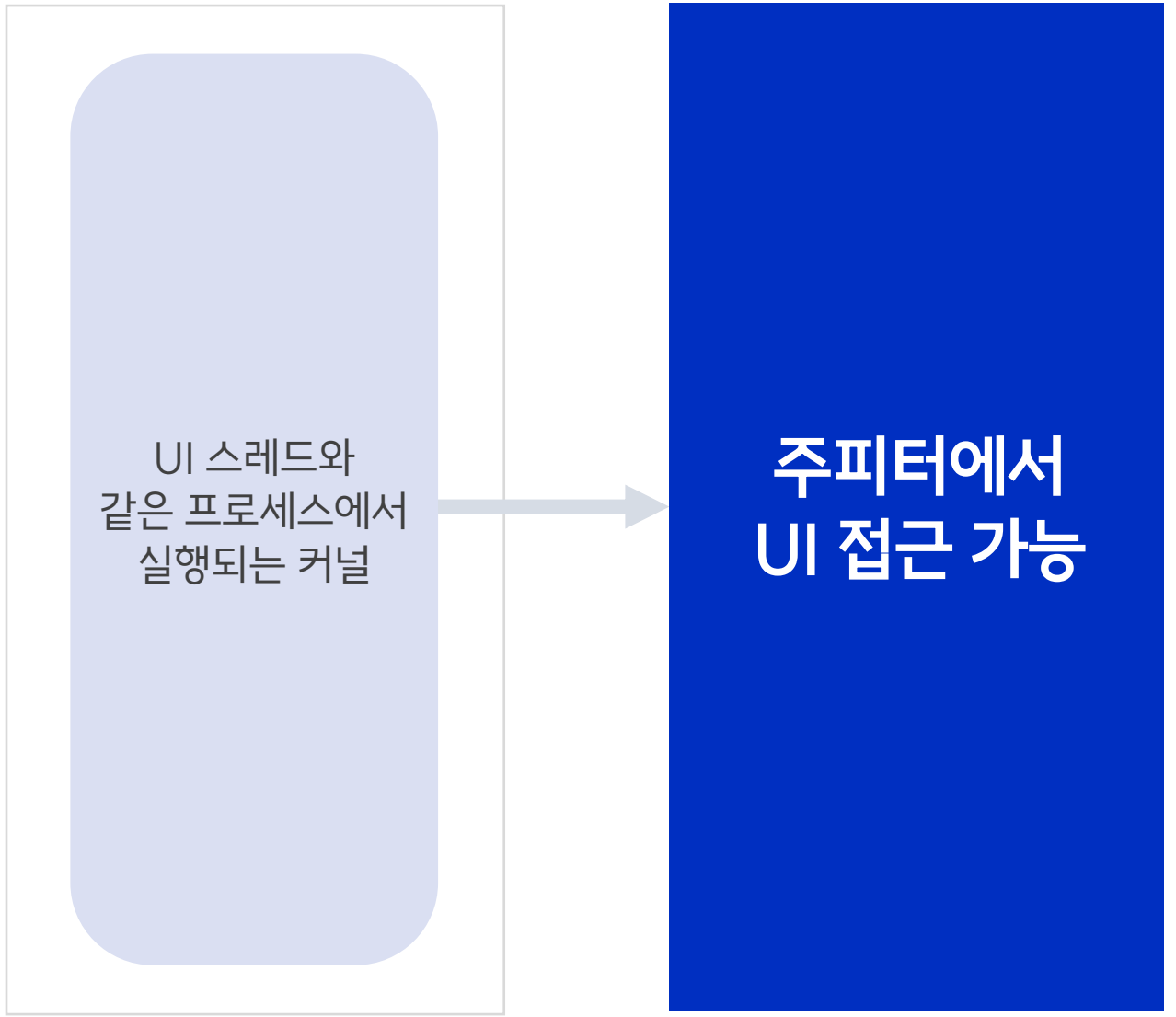
Part 3

멀티플랫폼 개발 보조 도구 (PythonX, PyREPL)



Part 3

멀티플랫폼 개발 보조 도구 (PythonX, PyREPL)



PyComposeUI PyREPL 데모에 오신 것을 환영합니다!
- Android Demo v0.0.1

[Jupyter Lab 연결 정보]
> IP: 192.168.219.103
> PORT: 55555
> PASSWORD: password

실행 모드 변경 (현재 옵션: UI 스레드 동기화 모드)

브라우저: 기본 브라우저 열기

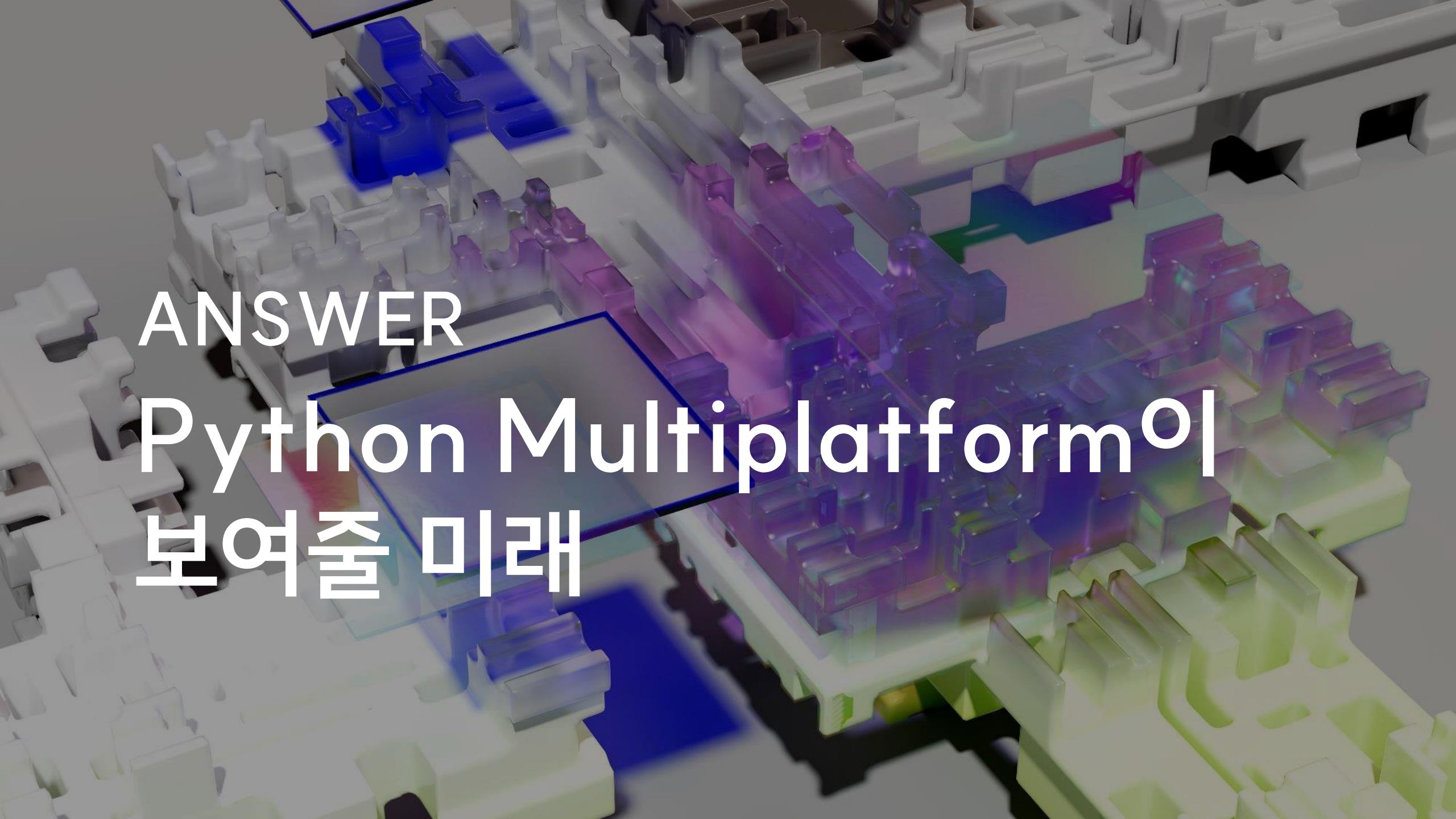
Jupyter Lab 실행



Part 4

기대효과 및 활용 분야

Usage/Applications



ANSWER

Python Multiplatform이 보여줄 미래

Part 4

기대효과 및 활용 분야

빠른 프로토타이핑

일반적으로, 모바일 앱 개발은 코드 한 줄을 수정하더라도, 수정 사항이 반영된 화면을 보기까지 **대기 시간이 존재할 수 밖에 없는데**, Python의 Jupyter Lab을 UI 스레드와 동기화 하여 실행하여 **UI 컴포넌트 수정을** 직접 주피터에서 **블록 단위로 수행**할 수 있으므로, 매우 빠르게 화면을 구성해서 볼 수 있다는 장점이 있음. (스타트업과 같은 시장에 매우 큰 장점)

온디바이스 인공지능 앱 개발

배포와 개발의 일원화: 온디바이스 인공지능 어플리케이션 개발에 있어, 개발 과정을 파이썬으로 일원화 할 수 있는 방안을 제시하기 때문에 개발 기간의 축소와 개발 과정상의 반복 작업을 일부 해소해줄 수 있다는 장점이 있음.

쉬운 데모: 또한, 인공지능 개발자들이 학습 완료한 모델을 쉽게 온디바이스에 올려 시연해볼 수 있기 때문에 개발 과정에서도 큰 이익이 있음.

앱 원격 업데이트

React-Native와 같이 **파이썬의 런타임 인터프리터를 앱 안에 내장**하였으므로, 런타임에 수정된 앱 코드 수신 및 JIT 컴파일이 가능하여 앱스토어의 허락 없이도 **원격 앱 업데이트**가 가능해짐. (스타트업과 같은 시장에 매우 큰 장점)

코틀린과 파이썬의 생태계 통합

라이브러리 상호 공유: 파이썬은 매우 방대한 생태계를 보유하고 있으나, 모바일 관련된 라이브러리는 이제 막 발전하기 시작한 단계이기 때문에, 코틀린과 파이썬의 생태계를 통합한다면 서로의 장점을 교환하는 윈-윈이 될 수 있음. (OS 제어는 코틀린, 인공지능은 파이썬)

코틀린 코드와 유연한 통합: 앱 일부를 코틀린으로 작성하고 나머지 일부를 파이썬으로 작성한 후 그를 통합하는 것이 매우 편리.

많은 개발자 유입: 파이썬을 사용하는 매우 많은 수의 개발자가 모바일 개발로도 유입될 수 있는 창구 역할 수행 가능.

프로젝트 3개년 계획


1차년도
1. Kotlin-Python 바인딩
2. Compose UI 랩핑
3. Juypyter 포팅
2차년도
1. Dart-Python 바인딩
2. Flutter UI 랩핑
3. Package Toolchain
3차년도
1. 성능 최적화
2. 경량화

Thank You



