Really Cross-Platform Python Development

PYCONFR 2023

Pascal Chambon, Freelance, **Witness Angel Project**

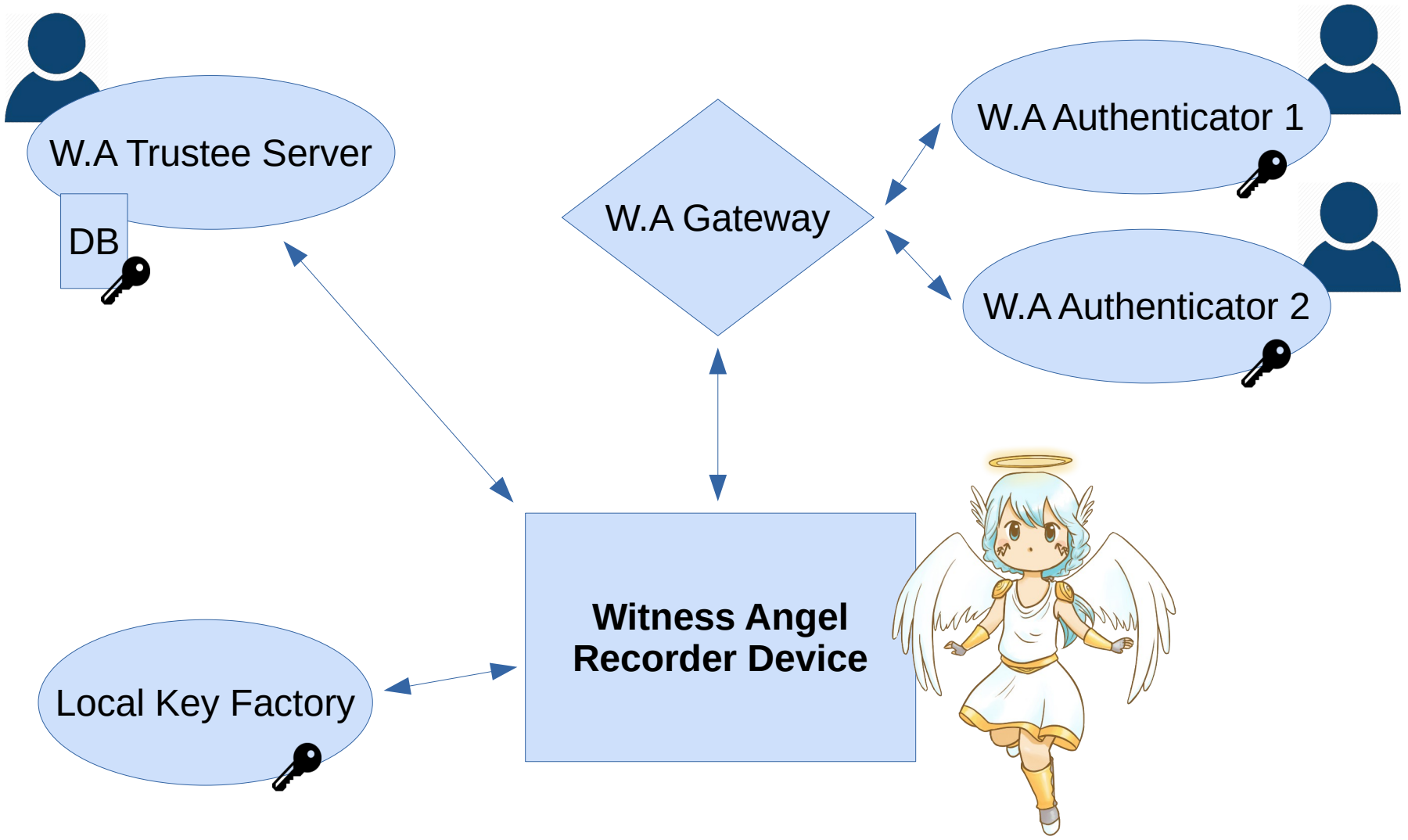
ONE DREAM

- Single components codebase
- Windows + Linux + Mac + RaspberryPi + iOS + Android
- Multi-architecture (x64+arm64)

Why we need it so much

- Witness Angel: tiny collective
- We develop “flights recorders“ for humans
- Use cases: harassment, abuse, aggression...
- Focus: Judicial Truth & Privacy
- OPPOSITE of Spy-Cams & Videosurveillance
- Neutral “Key Guardians“ to protect recordings

Witness Angel Ecosystem



I choose you... PYTHON

*Python is a *cross-platform*, interpreted, object-oriented programming language*

- Good for prototyping
- Good for webservices
- Good for cryptography
- Good for interoperability
- Good for… smartphones?

Let’s see if/how we can do it!

The Basics Of Interoperability

STEP 1 - Language

- We'll focus on **CPython**
- Lots of other implementations exist
  - PyPy, IronPython, Jython, GraalPython, Micropython…
  - Aligned with different CPython reference versions
  - Main roadblocks: C/C++ extension modules
- Usually, across different implementations, Python features behave *just the same*
  - Classes, metaclasses, properties, decorators, context managers, annotations, exceptions, comprehensions...

STEP 2 - Filesystem

Most blatant difference between Posix and Win32

Nope	Nice
Folder + “/” + File	Module pathlib, else os.path
“/tmp”, “C:\Windows\temp”	Module tempfile
str(path).lower()	Respect case sensitivity
os.chown(), os.chmod() unconditionally	Conditionally use os and pywin32 functions
“COM”, “NULL”, “:”, “;”, “/”...	Avoid reserved characters and words in file/folder paths

Story Time

A lib used urllib.parse.urljoin() for filepaths

Like… seriously.

Story Time

Kivy-ios has “com/” folder in its Git sources….

Unlucky

STEP 2 - Filesystem

- Lots of other little things to worry about…
- Newlines: LF (posix) vs CRLF (win32)
  - Glory to default "Universal Newlines" open() mode
  - Write LF newlines by default
- Encodings: UTF8 all the way
  - Thou shallt not **`open("readme.txt").read()`**
  - **`PYTHONUTF8`** environment variable to the rescue

STEP 3 – OS-Specific syscalls

System calls, signals, file descriptors...

Nope	Nice
os.fork()	Module multiprocessing
fcntl, ioctl	lockfile, RSFile... *(shameless plug)*
SIGCHLD, SIGPIPE, SIGKILL, SIGUSR*, *SIGBREAK*	SIGINT, SIGTERM
os.getuid(), os.getgid(), os.startfile() unconditionally	Conditionally use os or pywin32 functions

STEP 4 – Compiled extensions

- Compiled code depends on everything :
  - OS, hardware architecture, version of glibc/msvc runtime, debug mode or not, other .so/.dll libs, etc.
- Best case : python binary "wheels" exist somewhere for your target OS
- Worst case : compiling under Windows
  - Must have the proper Visual Studio version
  - "unable to find vcvarsall.bat"
  - "Cannot open include file: lber.h" (e.g. python-ldap)

OK, so now our algorithms and basic I/O work

UI & MEDIA

Graphics, Audio, Video

STEP 5 – GUI Toolkit

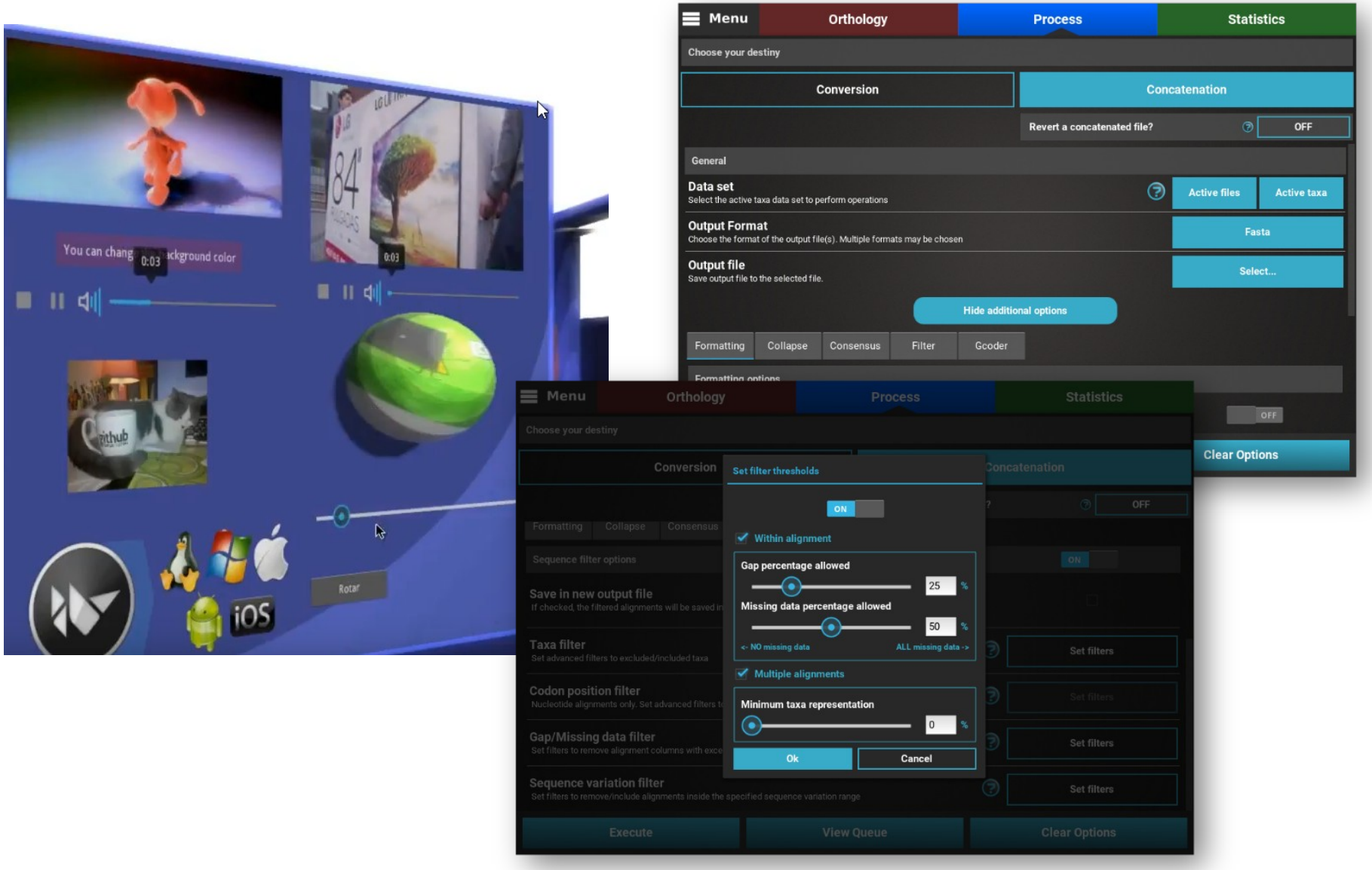
- Plenty of them for Python
  - Tkinter, WxPython, PyQt/PySide2, PyGObject...
- BUT... they only support "Desktop" systems
- A few pioneers exist nonetheless...

STEP 5 – GUI Toolkit

- Beeware (OS-native widgets)
  - Perspectives were unclear in 2020
  - Now doing steady progress on iOS/Android
  - Still small documentation and community feedback
- pyqtdeploy (PyQt widgets)
  - Has iOS and Android targets
  - Few tutorials
  - Very limited community feedback

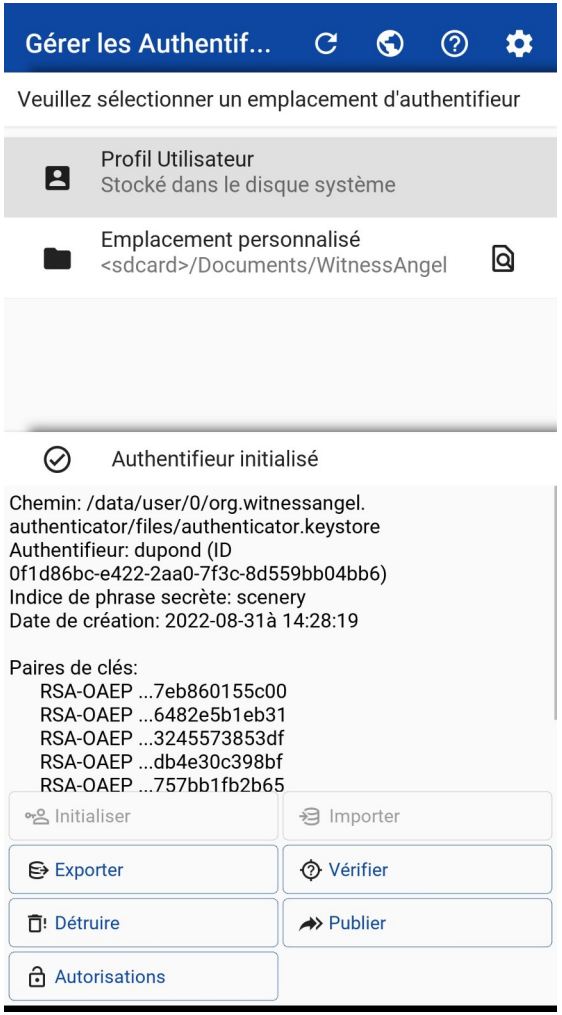
Doubts in front of the mountain...

Meet Kivy!


Menu
Orthology
Process
Statistics
Choose your destiny
Conversion
Concatenation
Revert a concatenated file?
OFF
General
Data set
Select the active taxa data set to perform operations
Active files
Active taxa
Output Format
Choose the format of the output file(s). Multiple formats may be chosen
Fasta
Output file
Save output file to the selected file.
Select...
Hide additional options
Formatting
Collapse
Consensus
Filter
Gcoder
Clear Options
Set filter thresholds
ON
Within alignment
Gap percentage allowed
25
Missing data percentage allowed
50
<- NO missing data
ALL missing data ->
Multiple alignments
Minimum taxa representation
0
Ok
Cancel
Taxa filter
Codon position filter
Gap/Missing data filter
Sequence variation filter
Set filters
Execute
View Queue
Clear Options
Rotar
iOS

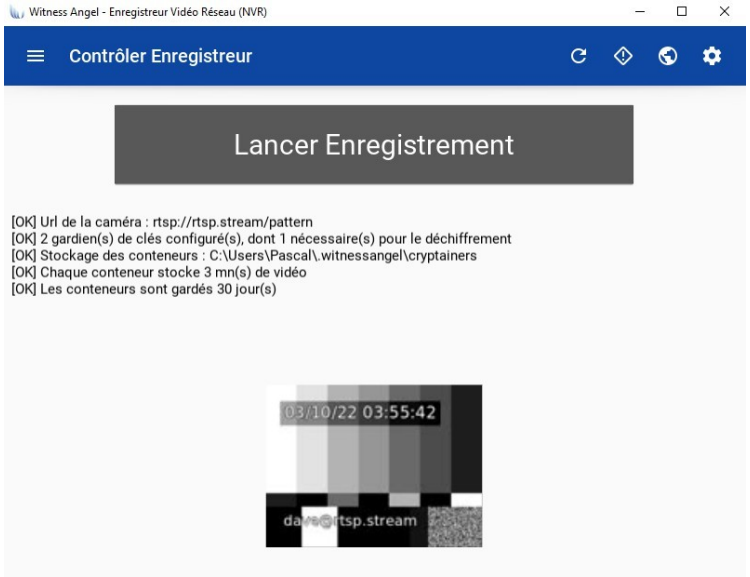
Kivy Principles

- Cross-platform
  - Windows/Linux/Mac/RaspberryPi/iOS/Android...
- Widgets based on OpenGL
  - 3D-enabled *(but no Accessibility yet)*
  - Video/audio added via SDL2/GStreamer
- Access to native mobile APIs
  - Via Pyjnius for Android, and Pyobjus for iOS
- Access to C/C++ libs on mobile
  - Via Android NDK, and Apple Xcode



A few liters of sweat later…

Time to distribute!



STEP 6 – Packaging

Lots of steps to "freeze" a Python program

- Bundle app's .pyc files
- Include pip-installed packages
- Include a python interpreter with its stdlib?
- Installable? Single-file? Self-extracted at runtime? Optimized? Compiled to binary?

Packaging for Desktop

- Lots of great options: Pyoxidize, Pyinstaller, cx_freeze, Briefcase, Nuitka...
- Plus some OS-specific: Py2exe, Py2app...
- Kivy provides helper **hooks** for Pyinstaller!
- Pyinstaller runtime workflow:
  - Run native "bootloader"
  - Launch Python interpreter
  - Load python modules (via ImportHooks)

Packaging for Desktop

Pyinstaller steps (Windows/Linux/Mac)

1) `$ Pyinstaller [--onefile] myscript.py`

2) `Tweak 'myscript.spec' file`

3) `$ Pyinstaller myscript.spec`

4) `Check the resulting folder/file`

Packaging for Desktop

Example spec file

```
als = Analysis(['minimal.py'],
               pathex=['/Developer/pylibs/'],
               Binaries=[...],
               Datas=[...],
               Hiddenimports=[...],
               runtime_hooks=[...])

pyz = PYZ(als.pure, als.zipped_data)

exe = EXE(pyz, ...)

coll = COLLECT(...)
```

Packaging for Desktop

Hints

- Gotta help Pyinstaller find some dependencies
- For compatibility, use oldest OS possible
- Beware of dependencies to system DLLs
  - Use the cleanest environment/VM possible
  - E.g. kivy-sdk-packager environments
- Antivirus software might dislike the magic of auto-extracted bundles

Packaging for Desktop

Some fun with Mac Silicon M1

- Architecture : not x64 but arm64
- Gotta install Rosetta2 for compatibility
- Gotta install two Homebrew environments
- Handy tools to survive the architecture mix
  - "lipo -info <executable>" to check a binary
  - "arch -x86_64/-arm64 <executable>" to force a mode
  - AppStore widgets to know which arch is running

Packaging for Desktop

OK, so in the end, we got:

- 1 ELF x64 executable for Linux
- 1 PE x64 executable for Windows
- 1 MACH-O x64+arm64 executable for Mac

Are we there yet?

Packaging for Desktop

Nope. Gotta sign executables for distribution

- Linux: no need, and don't even try
- Windows: optional ; standard SignTool will do
- Mac: now mandatory, fortunately Pyinstaller can help (with –codesign-identity)

From then on, any usual distribution channel (Installer, OS Store…) can be used.

Packaging for Desktop

Some more fun, with macOS Gatekeeper

- Gotta decide between exe, .app, .dmg, .pkg
- Gotta sign all levels of the package
- Gotta notarize (with altool) all components
- Better staple (with stapler) the "top package"
  - This helps app launch without Internet access

RASPBERRY PI

Raspberry Pi Peculiarities

- Different architectures
  - ARM v6 and v7 (32 bits), v8 (64 bits)
  - Official repos target ARM v6 for retrocompatibility
- Raspberry Pi OS ~= Linux
  - Pi Zero and its 512MB Ram are sluggish
  - Beware, 2 different camera stacks, mmal vs libcamera
- We configure+image it via Ansible
- World shortage of Raspberry Pis for now...

Smartphone Time!

Porting to Android and iOS


SOUNDS LIKE
HERESY TO ME

Mobile Operating Systems

A disconcerting context for “desktopers”

- Built-in sandboxing and permissions
- Some unusual technical limitations
- Additional limitations enforced by app stores
- Constant changes in APIs/Toolchains
- Simulators often use a different architecture

Running Python on Mobile

Principles

- A native bootloader initializes the process
- It spawns a cross-compiled Python interpreter
- Stdlib and app files are loaded from storage
- Java/Objective-C bridges to access devices

ANDROID

Android Peculiarities

- Multiprocessing still works
  - In Python, communicate e.g. via OSC protocol
- Normal permissions requested at install time
  - E.g. accessing network, bluetooth...
- Dangerous permissions requested at runtime
  - E.g. accessing shared folders, camera, location...
- Changes in GooglePlay packaging format
  - Previously “APKs”, now “Android App Bundles”

Porting to Android

- Check **`kivy.platform == "android"`**
- Use “android” module for permissions
- Use Plyer (wraps Pyjnius) for sensor access
- App booting could be a bit long
- Stdout will go to Android “logcat”
- Check files location with “adb shell”

Packaging for Android

Buildozer is all you need!

- Builds your package
  - Buildozer relies on Python-for-android toolchain
  - It installs/runs compilers (i.e. Gradle) for you
  - It relies on “recipes“ to compile complex Python packages
  - See buildozer.spec example in our repositories
- Deploys test app on ADB-connected device
- Signs and bundles app for production

Packaging for Android

Buildozer in practice

```
1) $ buildozer init   # Create buildozer.spec

2) Tweak buildozer.spec (script, logo, perms...)

3) $ buildozer android debug  # Build for debug

4) Plug your phone and setup ADB debugging

5) $ buildozer android deploy run logcat  # Run

6) $ buildozer android release # Build for prod
```

Packaging for Android

Distributing your app on PlayStore

- You need a Google developer account
  - One-time 25$ fee on enrollment
- You setup your Google Play Console account
- You upload/request signing keys
- You checkup app in closed/open beta tests
- You schedule app for verification and release

iOS

iOS Peculiarities

- Permissions are requested at install time only
  - No shared folder, expose public app folder instead
- Fork() is blocked on non-jailbroken devices
  - You’re limited to threads
- Dynamic libraries are disallowed by AppStore
  - BIG problem e.g. with Pycryptodome
- Interpreters are theoretically forbidden too
  - E.g. JVM… let’s be discrete for Python

Porting to iOS

- Check **`kivy.platform == "ios"`**
- Reuse Plyer (wraps Pyobjus) for sensor access
- App booting could be a bit long, still
- Export device logs via Xcode
  - For realtime logs : AppleConfigurator2, iTools, iOSLogInfo...
- Export “Package Contents” via Xcode
  - Useful to check the files pushed to app Sandbox

Packaging for iOS

- Buildozer is (imho) not operational
- Install Homebrew dependencies, and kivy-ios
- Use “toolchain” CLI
  - Cross-compile Kivy and dependencies with ”recipes”
  - This turns dynamic libs (.so) into static libs (.a)
  - Generate a Python-enabled Xcode project
- From then on, it’s a “normal” Xcode project
  - Full of settings, simulators, deployers etc.

Packaging for iOS

Distributing your app on AppStore

- You need an Apple Developer subscription
  - 99$ per year, darn...
- You setup your AppStore account and keys
- You upload app via Xcode
- You checkup app in beta tests (with TestFlight)
- You schedule app for verification and release

Packaging for iOS

Hints

- Apple validation is picky
  - Remove unwanted dynlibs from Xcode project
  - Explain why Kivy contains camera-access code
- TestFlight looks disruptive for OS
  - Friend lost all Instagram accounts when installing it

BONUS PLATFORMS

Micropython

A Python implementation for microcontrollers

- Like CPython3.4, with backported features
- Runs in hardware-constrained environments
- Subset of the stdlib, with specific modules
  - Bluetooth, cryptolib, network…
- Lots of CPython incompatibilities!
  - Multiple inheritance, module loading e

Micropython-ready Boards

Webassembly

Python in Browser, an old fantasy…
PyJS, Brython, Skulpt, Transcrypt...

- WASM, a new language-agnostic bytecode
  - Replacement for PNaCl, asm.js...
- Python3.11 documents support for WASM
  - On Emscripten SDK, inside browser
  - On WASI SDK, outside browser
  - See PyScript and Pyodide projects for quick starts
- Works quite fine (see FOSDEM talks)
  - Still some rough edges (app size, garbage collector...)

Wisdom Gained

- *Really Cross-platform Python* can be very complex, plan lots of debugging time
- Mobile ecosystem is risky
  - Because of small dev communities
  - Because of Google and Apple monopolies
- But you CAN have a single codebase
- Was it worth if for W.A? *Yes**1000*

Thanks for your attention!

- Feedbacks & contributions are welcome

  https://github.com/WitnessAngel/

- Website & Social Network

  https://witnessangel.com
  https://www.instagram.com/witnessangel_fr/

- Any questions?