A Python for Future Generations

Armin @mitsuhiko Ronacher
Hi, I'm Armin
...and I do Open Source,
lots of Python and SaaS
...Flask
Sentry...
... and here is where you can find me

twitter.com/@mitsuhiko
github.com/mitsuhiko
lucumr.pocoo.org/
‘raising awareness’
the grass is always greener somewhere
... what's Python anyway?
Python is whatever cPython does.
behavior & stdlib
a + b = ?
a.__add__(b) ?
type(a).__add__(a, b)?
a.__class__.__add__(a, b) ?
they are all not necessarily correct
LOAD_FAST 0 (a)
LOAD_FAST 1 (b)
BINARY_ADD
which is "obj as num". add
or "obj as sequence". concat
gave us unclear behavior when subclassing builtins
there is no "+" operator
there is PyNumber_Add
and PySequence_Concat
does it matter?
debatable but ... kinda?
because
pypy, jython all copy the quirks
because they want high compatibility
because

users would not use it if it was not compatible
prevents more innovative language changes
Python in 30 Years?
make the python we use more like the python we teach
it's a common story
python developers
value compatibility
distutils

implements original setup.py
setuptools

monkey patches distutils to support Python eggs
pip

monkey patches setuptools on the fly to manage python packages
monkey patches setuptools to build wheels instead of eggs
monkey patches setup tools and distutils to build extensions
snaek

monkey patches cffi to build Rust extension modules
the GIL
the only reason removing the GIL is hard is backwards compatibility
looks like we're not good at breaking compatibility
our only attempt was both radical and not radical enough
future of “scripting” languages
they are here to stay
but they will look different
standards + ecosystem
if we want to be here in 30 years, we need to evolve
where we did well
interpreter code is readable
ease of compilation
extensibility
flat dependency chains
runtime
introspection
what we should probably do
easier and clearer language behavior
looking elsewhere
JavaScript
Rust
both are new and modern
both learned from mistakes
packaging and modules
packaging and modules

package.json
Cargo.toml
packaging and modules

• metadata is runtime available
• by default no code execution on installation
• (optionally) multiple versions per library
• public vs private / peer dependencies
where are we now?

- we're moving away from `setup.py install`
- `pip` is a separate tool
- wheels
- multi-version would require metadata access
packaging and modules

realistic change?

- we can steal from others
- can target python 3 only if needed
language standard
language standard

- javascript: clarify interpreter behavior
- simplified language subset?
- generally leaner language?
- more oversight over language development
realistic change?

- maybe micropython and other things can lead the way
- community can kill extension modules for CFFI
unicode
utf-8 everywhere
wtf-8 where needed
• very little guessing
• rust: operating system string type
• rust: free from utf-8 to os-string and bytes
• explicit unicode character APIs
• emojis mean no basic plane
packaging and modules

realistic change?

- we would need to kill string slicing
- utf-8 everywhere is straightforward
- kill surrogate-escapes for a real os string?
extension modules
extension modules

more cffi
less libpython
extension modules

realistic change?

• tricky for things like numpy
• generally possible for many uses
linters & type annotations
linters & type annotations

babel, eslint, ...
typescript, flow, ...
linters & type annotations

rustfmt, gofmt, prettier, …
• maybe?
• typing in Python 3 might go this way
what you can do!
abuse the language less
sys._getframe(N).f_locals['_wat'] = 42
class X(dict):
stop writing non cffi extensions
stop being clever with sys.modules
awareness is the first step