Mojo

Python’s Typescript moment?

Team inovex
Karlsruhe · Köln · München · Hamburg
Berlin · Stuttgart · Pforzheim · Erlangen

inovex

Mojo

- Why Mojo?
- Mojo's Concepts
- Look at the Playground

inovex

Why Mojo?

Isn’t Python great?

Why Mojo?

- Unifying the world's ML/AI infrastructure
- Providing an innovative and scalable programming model for accelerators and other heterogeneous systems
- Addressing the limitations of existing languages (Python)

Source: https://docs.modular.com/mojo/why-mojo.html

Why Python?

- Dominant in ML and countless other fields
- Easy to learn, known, has community, packages, tooling
- Dynamic programming features support beautiful APIs
- Non-negotiable for Modular's API surface stack
- We believe Python is beautiful

or “during a gold rush, sell shovels” 🟨⛏️

Source: https://docs.modular.com/mojo/why-mojo.html

Why should be care?

Chris Lattner is the CEO of Modular

- Clang Compiler / LLVM
- Swift Programming Language

Tim Davis is the CPO

- LLVM / MLIR
- Google ML/Brain

Image Source: https://en.wikipedia.org/wiki/Chris_Lattner

(C)Python's shortcomings

- Poor low-level performance
- Single threaded (GIL)
- Building & Debugging C/C++ Extensions is complicated
- Old & problematic programming concepts
- Deployment

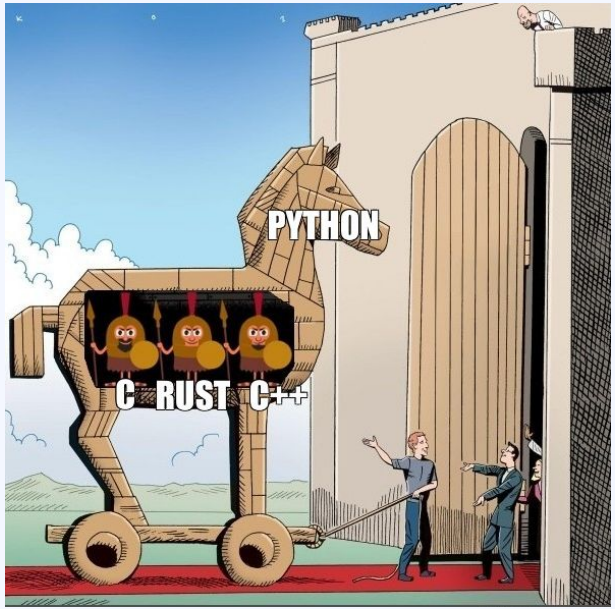

Image Source: https://www.meme-arsenal.com/en/create/meme/1094474

inovex

Mojo’s Concept

Will Mojo break the C/C++/Cuda monopoly?

`let` and `var` declarations

- 🔒 `let` is immutable
- 🔓 `var` is mutable
- 🎉 Lexical scoping
- 👥 Name shadowing
- ▯ Type specifiers

`struct` types

- Mojo equivalent of Python's `class`
- support methods, fields, operator overloading, decorators, ...
- bound at compile-time
- must be declared with `var` or `let`
- Additional build-ins structs
  - `Int`
  - `Float`
  - `StringLiteral` (`\0` terminated)
  - ...

Strong type checking

- **`struct`** type with compile-time-bound value specifications

```
def pairTest() -> Bool:
        let p = MyPair(1, 2)
        return p < 4 # gives a compile-time error
```

- Mojo supports type hints & strong type specifications
- Only code with strong types will allow the compiler to make more aggressive optimizations

`fn` - new function/method definitions

`fn`: strict mode of `def`

Feature	`fn`	`def`
Argument mutability	Immutable	Mutable
Argument type specification	Required	Optional
Return type specification	Required	Optional
Local variable declaration	Explicit	Implicit
Exception handling	Explicit	Implicit

Python `def` vs Mojo `def` vs Mojo `fn`

Feature	Python def	Mojo def	Mojo fn
Argument passing method	Reference semantics	Value semantics by default	Immutable references by default
Mutability of arguments	Mutable	Mutable	Immutable by default
Visibility of changes to arguments	Visible outside the function	Not visible outside the function	**`borrowed`**: Arguments cannot be changed **`inout`**: visible outside the function
Argument passing convention		Copy (**`__copyinit__`**)	Borrowing

Borrow Checker 🦀

- There can only be one mutable reference to the same value
- Multiple immutable borrows per value are possible
- Cannot pass one mutable (`inout`) and one/more immutable (`borrowed`) references at the same time
- Mojo `fn` arguments borrow by default (think C++ `const&`)
- Small values (`Int`, `Float`, `SIMD`) are passed directly in machine registers
- No sigils (`&`) needed to pass as immutable borrowed reference

Transfer arguments (`owned` and `^`)

- **`owned`** argument convention is for functions that take exclusive ownership
- The **`^`** operator transfers ownership of a value to another entity

```
fn take_ptr(owned p: SomeUniquePtr):
    use(p)

fn usePointer():
    let ptr = SomeUniquePtr(...)
    use(ptr)        # Perfectly fine to pass to borrowing function.
    take_ptr(ptr^)  # Pass ownership of the `ptr` value to another function.

    use(ptr) # ERROR: ptr is no longer valid here!
```

Python integration

- **`Python.import_module()`** imports a module into Mojo (Importing individual members is not yet available)
- **`Python.add_to_path()`** to add local Python code
- Memory management works out of the box
- Mojo primitive types implicitly convert into Python objects

Python integration

my-python.py:

```python
import numpy as np

def my_algorithm(a, b):
    array_a = np.random.rand(a, a)
    return array_a + b
```

mojo-code.mojo:

```
from PythonInterface import Python

Python.add_to_path("path/to/module")
let py = Python.import_module("my-python")

let c = py.my_algorithm(2, 3)
print(c)
```


inovex

Interlude: Multi-Level Intermediate Representation (MLIR)

- Reusability: MLIR can be used to create compilers for a variety of languages and hardware platforms.
- Extensibility: MLIR is designed to be extensible, making it easy to add new features and optimizations.
- Flexibility: MLIR supports multiple levels of abstraction, making it well-suited for a variety of compiler applications.

```
func @add(x: f32, y: f32) -> f32 {
  %add = addf x y
  return %add
}
```

See https://mlir.llvm.org

MLIR in Mojo

- Target multiple accelerators
  - GPUs
  - TPUs
  - CPUs
  - ...
- Built In auto-tuning and adaptive compilation (vector-length-agnostic algorithms)
- Low-level IR in Mojo code

```
struct OurBool:
    var value: __mlir_type.i1
```

 See https://docs.modular.com/mojo/notebooks/BoolMLIR.html

Playground

https://playground.modular.com/

inovex

Mojo Resources

🔥 https://www.modular.com/mojo

🦮 https://www.modular.com/get-started

 https://docs.modular.com/mojo/

🎥 Modular Product Launch 2023 Keynote

🎥 Fireship: Mojo Lang... a fast futuristic Python alternative

Vielen Dank!

Bernd Kaiser

Software Developer

bernd.kaiser@inovex.de

inovex