Intro to python, assembly, c/c++, java, and the removal of compilers:

By Dylan Eliot

my Email -- dylan.k.eliot@gmail.com

my github by username -- https://www.github.com/asmeble

512-361-6942 for phone calls & texts

Who am I?

c/c++ developer
bash & make script editor
python developer
platform & kernel developer
assembly programmer
web developer

What is the difference between a compiler and an interpreter?

A compiler needs to translate what you've given it into a context the machine understands before running.

Assembly is required and you have to build the batteries.....

An interpreter just needs code to run & needs assembly to be run on specific hardware

Why were compilers needed?

How do they differ from python?

compilers were needed to abstract layers between hardware & software while maintaining cohesion. The compiler is meant to translate symbolic notation into algorithmic layout. The compiler knows assembly & the language symbols you're attempting to use, and at best will try to give a notation that seems to have a cohesive execution stream. Compilers only are aware of two scopes, and how to encode your information into something that can be executed.

c/c++ has a few requirements:

*requires a namespace for certain parts of it to run
*to be compiled to use ("compiled")
*an execution environment
*a "kernel" (poor torvald and c/c++ developers)
*a language it must be translated into
*assembly, kernel interrupt handlers, kernel functions, and a binary interpreter
*a foreign function interface (even for binaries)
*left and right are as subjective as up and down or top and bottom

Python is an interpreter. It required these things to run in an execution environment:

*repeatable
*semi-predictable
*based on c/c++
*can support c/c++ types & functions (hand code native support for c/c++ functions)
*can use unicorn-engine to support other architectures
*can load resources from across the web into native runtime
*can garbage collect on __exit__ as a way to prevent bloating of runtime
*type specificity is optional and assigned on the right side on the equation
*read from the top moving downward, read from left to right
*no namespace declaration required unless defining a class

What should be done instead of compiling?

emulation of the subsection you want to run like compiled code and compile only where necessary.....
Emulate as much of the compilable language as one can. Cache and memoize code as much as possible. Annotate my the token and where it should repeat as a sequence of dictionary values; the assembly code as keys, and where those keys are supposed to be in reference to the position in "memory".

Wouldn't emulating and simulating another language complicate the complexity of interpreting?

Why would it if all the instructions that take longer are removed and substituted for operations that take less time to process?
Why would such mess with interpreting a language, and how would it impede that portion of the process of execution and interpretation of another?

Is an all python runtime possible?

No, but a mostly python one is likely.
https://github.com/Asmeble/Pure_python_kernels

The content contains information relevant to making any operating system use a emulation of c/c++ functions inside of python. Their are kernel functions as well as kernel interrupt handlers. This allows for one to define an OS any way they want.

The next link is for BIOSBITS:
https://github.com/biosbits

The developer abstracted away from the OS into grub, a linux like bootloader, through modular design.
The overall arch his project made was that you could touch hardware and probe it at a lower level.
If you wanted metal to describe a mandelbrot set, you could with the write RGB values directly to the display.

What does this mean for compiling code?

It will slowly come to an end, where a few things are compiled and most of it is emulated. It should function with little to no error.

If anything, It may mean that the compiling phase is removed completely from the equation.