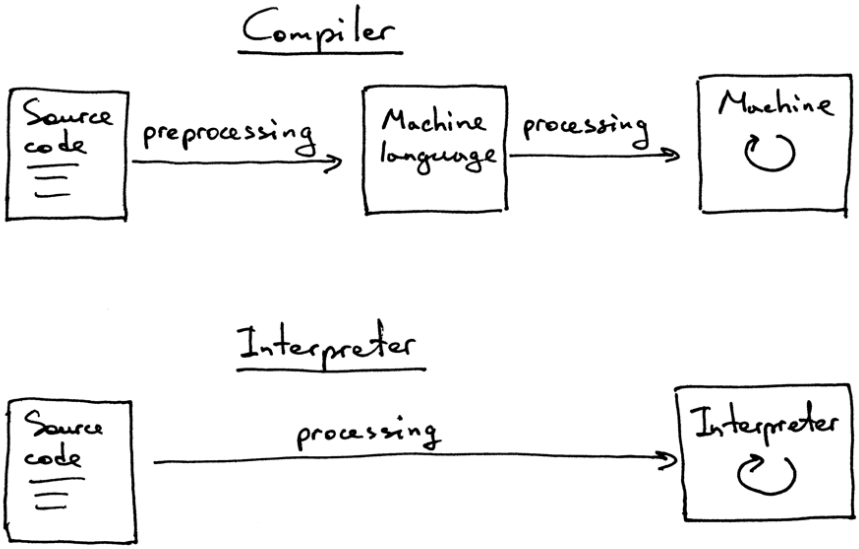
How does python really works?



before we move on let talk about what is an interpreter and how it's different from a compiler.

Compiled vs Interpreted.

In a compiled implementation, the original program is translated into native machine instructions, which are executed directly by the hardware.

In an interpreted implementation, the original program is translated into something else. Another program, called "the interpreter", then examines "something else" and performs whatever actions are called for. Depending on the language and its implementation, there are a variety of forms of "something else"

What's about python?

python has its own virtual machines. this VM is more likely JVM and not like a virtualbox. It's CPU that build with a complex software that can executed "bytes code" that compiled by python compilers.

What happen when you hit return key?

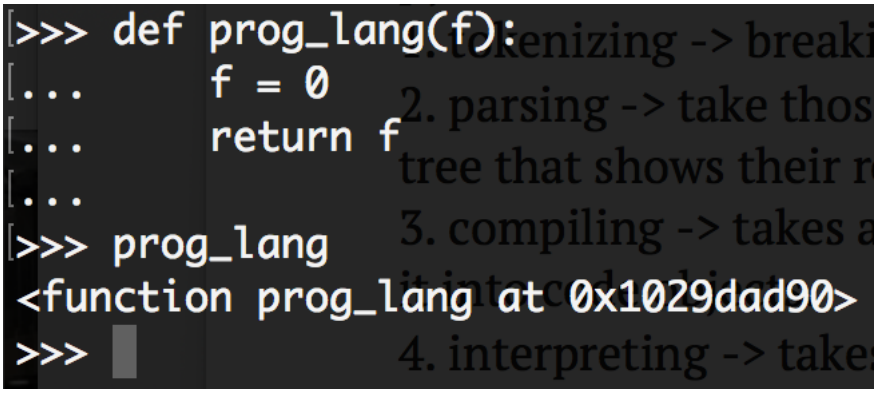
there are 4 steps that python take

1. **tokenizing** -> breaking the line of code into tokens.
2. **parsing** -> take those tokens and generates a syntax tree that shows their relationship to each other.
3. **compiling** -> takes a tree from parsing phase and turn it into code objects.
4. **interpreting** -> takes each "code object" and executes the code.

Function are objects in python.

in python functions are objects, like a list is an object. this allow we to pass a function into another function as an argument.

Ex. foo1 = foo2



What is "code object"?

A code object is generated by the Python compiler and interpreted by the interpreter. It contains information that this interpreter needs to do its job.

What is "bytecode"?

```
1  >>> [ord(b) for b in foo.func_code.co_code]
2  [100, 1, 0, 125, 1, 0, 124, 1, 0, 124, 0, 0, 23, 83]
```

bytecode is an attribute of the code object, among many other attributes. The interpreter will read through each byte and look up what it should do for each one.

Disassembling bytecode

bytecode is analogous to machine code -> Assembly so we can do the same thing we can disassembling it.

```
>>> def foo(a):
...     x = 3
...     return x + a
...
>>> import dis
>>> dis.dis(foo.func_code)
  2           0 LOAD_CONST               1 (3)
              3 STORE_FAST               1 (x)

  3           6 LOAD_FAST                1 (x)
              9 LOAD_FAST                0 (a)
             12 BINARY_ADD
             13 RETURN_VALUE
```

these assembly look like code is the code for abstract software CPU that create in python interpreting phase.

Conclusion

python compiler takes your code and "split in to tokens" then "make an syntax tree" and "compiled to a byte codes" then "interpreting and make an execution in virtual machines" so that make python be a "dynamic programming languages" because instructions didn't execute in a piece of hardware.

* **Dynamic programming language** is a class of high-level programming languages which, at runtime, execute many common programming behaviors that static programming languages perform during compilation.