The target of a method invocation like `o.m()` in statically typed languages such as Java, C++ and Scala is found in constant time: two loads, and then we have the address of the method `m`. In dynamically compiled languages such as Python, JavaScript and Ruby, finding the target of `o.m()` can take much longer!

```python
def fill(set, b, e, s):
  for i in range(b, e, s):
    set.add(i)
```

Figure 1: A python program.

1. What is the type of variable `set` in the Python code on the left?

UFMG DCC DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
Compilers Laboratory

2. Languages like Java and C++ use virtual tables to quickly find the target of a method invocation. What is a virtual table?

3. Why can't virtual tables be used in dynamically typed languages such as Python and JavaScript? (Well, not easily, at least?)

4. Nevertheless, languages such as Python and JavaScript still can find the target of method calls in amortized O(1) time. How would that be possible?

5. Your answer to question (4) might have relied on the fact that code for these languages can be generated on-the-fly (through Just-in-Time compilation). Would it be possible to find the target of a method call in a language like Python in constant time, in general, if the code were produced ahead of time?

6. These languages, Python, JavaScript, Ruby, Php, etc, are typically interpreted, and sometimes they can be compiled just-in-time. Could these languages be compiled ahead-of-time? Or, in other words, what makes compiling these very dynamic languages so challenging?