Session 1: What is computation?

Duy Nguyen

April 2, 2019

A computer does two things:

- Perform calculations.
- Remember the results of calculations.

Two types of knowledge:

- Declarative knowledge is composed of statements of fact, e.g., $\sqrt{9} = 3$.
- Imperative knowledge is "how to" knowledge, or recipes for deducing information, e.g., Babylonian method or Heron method for finding square roots (This is an example of a guess-and-check algorithm).

Algorimth definition:

- A sequence of simple steps, together with a flow of control that specifies when each step is to be executed.
- A finite list of instructions that describe a computation that when executed on a set of inputs will proceed through a set of well-defined states and eventually produce an output.

Two types of computer:

- Fixed-program computers were designed to do very specific things, e.g., a simple handheld calculator, Alan Turing's bombe machine.
- Stored-program computers stores (and manipulates) a sequence of instructions, and has components that will execute any instruction in that sequence, e.g., the Manchester Mark 1 - the first truly modern computer.

An interpreter is a program that can execute any legal set of instructions.

Flow of control is an order in which instructions are executed.

A programming language is a way to give the computer its marching orders.

Universal Turing Machine has an unbounded memory in the form of "tape" on which one could write zeroes and ones, and some very simple primitive instructions for moving, reading, and writing to the tape.

The Church-Turing thesis states that if a function is computable, a Turing Machine can be programmed to compute it.

Halting problem: Turing proved that it is impossible to write a program that given an arbitrary program, call it P, prints `true` if and only if P will run forever.

A programming language is said to be Turing complete if it can be used to simulate a universal Turing Machine.

Each programming language has:

- Primitive constructs: Literals (e.g., the number `3.2` and the string `'abc'`) and infix operators (e.g., `+` and `/`) in Python.

- The syntax of a language defines which strings of characters and symbols are well formed (e.g., in Python, the sequence of primitives `3.2 + 3.2` is syntactically well formed, but the sequence `3.2 3.2` is not).

- The static semantics defines which syntactically valid strings have a meaning (e.g., in Python, the sequence `3.2/'abc'` is syntactically well formed (<literal> <operator> <literal>), but produces a static semantic error since it is not meaningful to divide a number by a string of characters).

- The semantics of a language associates a meaning with each syntactically correct string of symbols that has no static semantic errors. In natural languages, the semantics of a sentence can be ambiguous. Programming languages are designed so that each legal program has exactly one meaning.

Every serious programming language does a complete job of detecting syntactic errors.

Some programming languages, e.g., Java, do a lot of static semantic checking before allowing a program to be executed. Others, e.g., C and Python (alas), do relatively less static semantic checking before a program is executed.

If a program has no syntactic errors and no static semantic errors, it has a meaning, i.e., it has semantics. Of course, that isn't to say that it has the semantics that its creator intended it to have.

Program error behaviors:

- It might crash.
- Or it might run forever.
- Or it might run to completion and produce an incorrect answer.

Programming language differences:

- *Low-level vs high-level* refers to whether we program using instructions and data objects at the level of the machine or whether we program using more abstract operations.
- *General vs targeted to an application domain* refers to whether the primitive operations of the programming language are widely applicable or are finetuned to a domain.
- *Interpreted vs compiled* refers to whether source code is executed directly (by an interpreter) or whether it is first converted (by a compiler) into machine code.

Python is a general-purpose programming language that can be used effectively to build almost any kind of program that does not need direct access to the computer's hardware.

Python advantages and disadvantages:

- Advantages:
  - It is a relatively simple language.
  - It can provide the kind of runtime feedback.
  - There are also a large number of freely available libraries that interface to Python.
- Disadvantages:
  - It has weak static semantic checking.

A Python program, sometimes called a script, is a sequence of definitions and commands.

These definitions are evaluated and the commands are executed by the Python interpreter in something called the shell.

A command, often called a statement, instructs the interpreter to do something.

Objects are the core things that Python programs manipulate.

Every object has a type that defines the kinds of things that programs can do with that object.

Types are either scalar or non-scalar.

- Scalar objects are indivisible.
- Non-scalar objects have internal structure.

Objects and operators can be combined to form expressions, each of which evaluates to an object of some type.

Variables provide a way to associate names with objects. An object can have one, more than one, or no name associated with it.

References

[1] Guttag, John. *Introduction to Computation and Programming Using Python: With Application to Understanding Data Second Edition.* MIT Press, 2016.

[2] `https://github.com/kaizenflow/6.0001-Notes`