# CSE 331: Reasoning

A programmer is someone who can produce code that they *know* works.

Programmers use three techniques to gain knowledge about their code:

- *Type checking*: The compiler can inform us of ways in which our code is inconsistent.
  - This technique is also known as static analysis. "Static" means "does not run the code".
  - Type checking considers *all possible executions* of the program. If the type checker passes, it guarantees the code is free from run-time type errors.
  - Type checking is necessarily imprecise. In particular, it does not have access to complete specifications, so it cannot tell us if our code is correct on all executions.
- *Testing*: We can manually or programmatically run the code and check that the output is correct on specific inputs.
  - Testing necessarily only considers the specific executions exercised by the inputs we chose to test.
  - If we derive our testing expectations from the specification, then testing can tell us that the program was correct *on the inputs we tested*.
- *Reasoning*: We can think carefully about what assertions hold at what points in our program and convince ourselves that the program is correct.
  - Reasoning considers all possible executions, and reasoning considers the full specification of the program.
  - Reasoning requires a programmer to think carefully.

An **assertion** is a true/false claim about the values of the variables of a program. Examples: `x` is positive, array `a` is sorted.

An assertion $Q$ **holds** at a program point $\ell$, if $Q$ is true on all executions of the program that reach $\ell$.

- An assertion can hold at some program points but not hold at other program points!
- The question of whether an assertion holds or not is a *global* one: it depends on all executions of the program.

**Global reasoning** means determining whether an assertion holds at a particular program point by thinking carefully about all executions.

**Local reasoning** uses a method's specification to break up the work:

- Reason about the method's implementation in isolation: assume the precondition at the top, convince ourselves the postcondition holds at the bottom.
- At each call site: convince ourselves the precondition holds, then "skip over" the method call by assuming the postcondition.

**Forward reasoning** refers to transforming known facts before a statement into known facts after the statement.

```
// Prints absolute value of first arg
public static void main(String[] args) {
    if (args.length == 0) {
        return;
    }
    // L1
    int n = Integer.parseInt(args[0]);
    // L2
    if (n < 0) {
        n = -n;
    }
    // L3
    System.out.println(n);
}
```

---

```
// Prints absolute value of first arg
public static void main(String[] args) {
    if (args.length == 0) {
        return;
    }
    // L1
    int n = Integer.parseInt(args[0]);
    // L2
    System.out.println(abs(n));
}
// @returns the absolute value of x
private static int abs(int x) {
    if (x < 0) {
        return -x;
    } else {
        return x;
    }
}
```