National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2016/2017

# Recitation 1
# Functional Abstraction

## Source

1. Source Week 3 allows for `true` and `false` as primitive expressions. These expressions evaluate to *boolean values* that can be used to take yes/no decisions.

2. Source Week 3 provides for the following comparison operators for numbers: `>`, `<`, `>=`, `<=`, `===` and `!==`. They work as expected: You can place them between two expressions. When such a comparison is evaluated, the two expressions are evaluated first. When they evaluate to numbers, the comparison evaluates to either `true` or `false`, depending on the numbers and the operator. For example, executing the program

   ```
   1 + 2 < 5;
   ```

   results in `true`.

3. Conditional statements of the form

   ```
   if (test-1) {
       cons-stmt-1
   } else if (test-2) {
       cons-stmt-2
   } else if (test-3) {
       cons-stmt-3
   } else {
       alt-stmt
   }
   ```

   evaluate a series of tests in order. If a test evaluates to `true`, the corresponding consequent is evaluated, otherwise the next test is evaluated. If a test evaluates to `true`, succeeding tests will not be evaluated. If none of the tests evaluate to `true`, the final alternative is evaluated.

   Example:

   ```
   function sign(x) {
       if (x < 0) {
           return -1;
       } else if (x > 0) {
           return 1;
       } else {
           return 0;
       }
   }
   ```

4. Similarly, conditional expressions of the form

   ```
   (test-1) ?
    consequent-expr-1 : (test-2) ?
                        consequent-expr-2 : (test-3) ?
                                            consequent-expr-1 : alt-expr
   ```

   evaluate a series of tests in order. If the value of a test is *not false*, the value of whole conditional expression is the value corresponding consequent, otherwise the next test is performed. In the former case, succeeding tests will no longer be evaluated. If all tests fail, the value of the whole conditional expression is the value of the remaining *alternative*.

   Example: The function above can be re-written as:

   ```
   function sign(x) {
       return (x < 0) ?
              -1 : (x > 0) ?
                   1 : 0;
   }
   ```

   Note that in Source, there must not be any newline character between the `return` keyword and the expression.

5. Function expressions of the form

   **function**(*parameters*){ *body* }

   create a function with the given parameters and body. *Parameters* is a comma-separated sequence of names of variables. *Body* is a Source statement. When the function is applied, the body statement is executed. The function can return a value to the caller using `return`, followed by an expression.

## Source IDE

1. The Source IDE provides an evironment for writing and testing Source programs.
2. Go to the Source IDE at `http://source-ide.surge.sh/`.
3. Click on the "Interpreter" under the "Source Week 3" playground to open an interpreter console window.
4. In the console window, you can enter Source programs, and press "Shift-Enter" to let the interpreter evaluate your program. The result of the evaluation is displayed below your program.
5. This feature of the Source IDE is a "read-eval-print loop" (REPL). It reads the input, executes (evaluates) the input as a Source program, and prints the result.

## Problems:

1. Use the interpreter console to evaluate the following statements, assuming `x` is bound to 3, and observe their effect:

```
if (true) { 1+1; } else { 17; }

if (false) { false; } else { 42; }

if (x > 0) { x; } else { -x; }

if (x === 0) { 1; } else { 2; }

if (x < 0) { 7; } else { 7; }

if (true) { 1; }
else if (y < 1) { 4711; }
else{ 42; }
```

2. Evaluate the following statements:

```
(function(x) { return x; });

(function(x) { return x; })(17);

(function(x, y) { return x; })(42, 17);

(function(x, y) { return y; })(z, 3);

(function(x, y) { return x(y, 3); })((function(a, b) { return a + b; }), 14);
```