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

Source
Week 4

Introduction

The language Source is the official language of CS1101S. You have never heard of Source? No wonder, because we invented it just for the purpose of this module. Source is a sublanguage of ECMAScript 2016 (7^{th} Edition) and defined in the documents titled "Source Week x". More specifically, the missions, side quests, competitions, practical, midterm and final assessments use the Source language of the current week or the next week x for which a document "Source Week x" is available.

Changes

Compared to Source Week 3, Source Week 4 has the following changes:

- A single semicolon is allowed as statement
- Functions are allowed to have no return statement (they return `undefined`)
- All `Math` operators are allowed, see `http://bclary.com/2004/11/07/` (Section 15.8)
- Curly braces around conditionals in `else` statement can be omitted
- `display(value)` can be used to display a value in the console

Statements

A Source program is a statement. Statements are defined using Backus-Naur Form (BNF) as follows:

⟨*statement*⟩ ::= `;`
| `var` ⟨*id*⟩ `=` ⟨*expression*⟩ `;`
| ⟨*if-statement*⟩
| `function` ⟨*id*⟩ `(` ⟨*id-list*⟩ `)` `{` ⟨*statement*⟩`}`
| ⟨*statement*⟩ ⟨*statement*⟩
| `return` ⟨*expression*⟩ `;`

	|	⟨*expression*⟩ `;`
⟨*if-statement*⟩	::=	`if (` ⟨*expression*⟩ `) {` ⟨*statement*⟩ `} else {` ⟨*statement*⟩ `}`
	|	`if (` ⟨*expression*⟩ `) {` ⟨*statement*⟩ `} else` ⟨*if-statement*⟩
⟨*id-list*⟩	::=	
	|	⟨*non-empty-id-list*⟩
⟨*non-empty-id-list*⟩	::=	⟨*id*⟩
	|	⟨*id*⟩ `,` ⟨*non-empty-id-list*⟩

Important note: There cannot be any newline character between `return` and ⟨*expression*⟩ `;`.

⟨*expression*⟩	::=	⟨*number*⟩
	|	`true` | `false`
	|	⟨*string*⟩
	|	⟨*expression*⟩ ⟨*bin-op*⟩ ⟨*expression*⟩
	|	⟨*un-op*⟩ ⟨*expression*⟩
	|	`function (` ⟨*id-list*⟩ `) {` ⟨*statement*⟩`}`
	|	⟨*id*⟩ `(` ⟨*expr-list*⟩ `)`
	|	`(` ⟨*expression*⟩ `) (` ⟨*expr-list*⟩ `)`
	|	⟨*expression*⟩ `?` ⟨*expression*⟩ `:` ⟨*expression* ⟩
	|	`(` ⟨*expression*⟩ `)`
⟨*bin-op*⟩	::=	`+` | `-` | `*` | `/` | `%` | `===` | `!==` | `>` | `<` | `>=` | `<=` | `&&` | `||`
⟨*un-op*⟩	::=	`!` | `-`
⟨*expr-list*⟩	::=	
	|	⟨*non-empty-expr-list*⟩
⟨*non-empty-expr-list*⟩	::=	⟨*expression*⟩
	|	⟨*expression*⟩ `,` ⟨*non-empty-expr-list*⟩

Identifiers

Variables in Source are syntactically represented by identifiers. In Source, an identifier consists of digits (0,...,9), the underline character _ and letters (a,...z,A,...Z) and begins with a letter or the underline character.

Builtin Functionality

The following identifiers can be used, in addition to identifiers that are declared using `var` and `function`:

- `alert(string)`: Pops up a window that displays the string
- `display(value)`: Displays a value in the console
- `Math.`⟨*name*⟩

where ⟨*name*⟩ is any name specified in the JavaScript `Math` library, see `http://www.ecma-international.org/publications/files/ecma-st/ECMA-262.pdf` (PDF) pages 159 and following, and `http://bclary.com/2004/11/07/` (HTML). Examples:

- `Math.E`: Refers to the mathematical constant e,
- `Math.PI`: Refers to the mathematical constant π,
- `Math.sqrt`: Refers to the square root function.

Note that technically, `Math.`⟨*name*⟩ is not an identifier, but more similar to an operator combination, the operator being "`.`". We will learn more about this construct when learning about objects.

Numbers

Examples for numbers are `5432`, `-5432.109`, and `-43.21e-45`.

Strings

Strings are of the form `"`⟨*characters*⟩`"`, where the character `"` does not appear in ⟨*characters*⟩, and of the form `'`⟨*characters*⟩`'`, where the character `'` does not appear in ⟨*characters*⟩.

Typing

Expressions evaluate to numbers, boolean values, strings or function values.

Only function values can be applied using the syntax:

⟨*expression*⟩ ::= ⟨*id*⟩ `(` ⟨*expr-list*⟩ `)`
| `(` ⟨*expression*⟩ `)` `(` ⟨*expr-list*⟩ `)`

The following table specifies what arguments Source's operators take and what results they return.

operator	argument 1	argument 2	result
`+`	number	number	number
`+`	string	any	string
`+`	any	string	string
`-`	number	number	number
`*`	number	number	number
`/`	number	number	number
`%`	number	number	number
`===`	number	number	bool
`===`	bool	bool	bool
`===`	string	string	bool
`===`	function	function	bool
`!==`	number	number	bool
`!==`	bool	bool	bool
`!==`	string	string	bool
`!==`	function	function	bool
`>`	number	number	bool
`<`	number	number	bool
`>=`	number	number	bool
`<=`	number	number	bool
`&&`	bool	bool	bool
`||`	bool	bool	bool
`!`	bool		bool
`-`	number		number

Following `if` and preceding `?`, Source only allows boolean expressions.

Comments

In Source, any sequence of characters between "`/*`" and the next "`*/`" is ignored.

After "`//`" any characters until the next newline character is ignored.

Remarks

In Source, the order of declaration of functions and variables matter. Always place function statements before all other statements. `var` actually declares variables to be usable in the entire function.