Client side scripting – Javascript

Javascript

- Javascript is a dynamic computer programming language.
- It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages.
- It is an interpreted programming language with object-oriented capabilities.

Javascript

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

Javascript

```
<html>
<body>
<script >
  document.write("Hello World!")
 </script>
 </body>
 </html>
```

Javascript

The most preferred ways to include JavaScript in an HTML file are as follows –

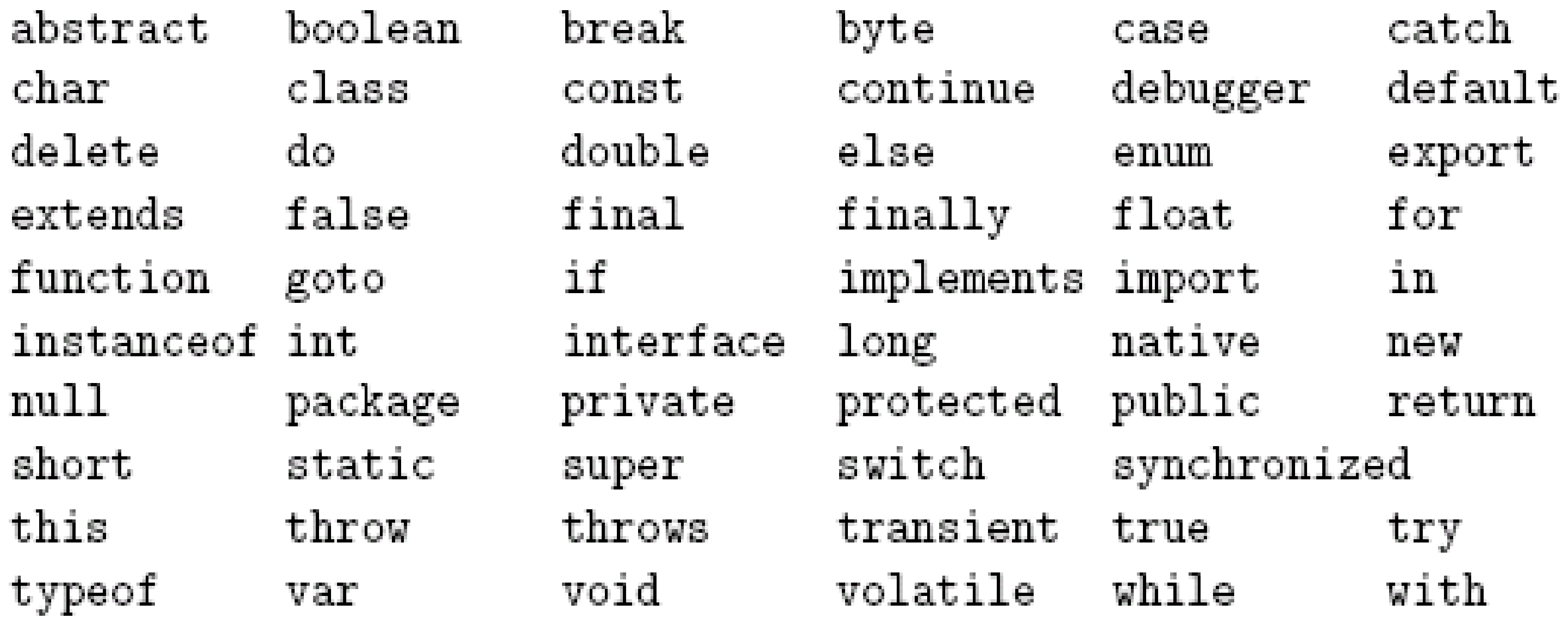
- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

Data types

JavaScript allows you to work with three primitive data types –

- **Numbers,** eg. 123, 120.50 etc.
- **Strings** of text e.g. "This text string" etc.
- **Boolean** e.g. true or false.
- Object

Reserved words



abstract	boolean	break	byte	case	catch
char	class	const	continue	debugger	default
delete	do	double	else	enum	export
extends	false	final	finally	float	for
function	goto	if	implements	import	in
instanceof	int	interface	long	native	new
null	package	private	protected	public	return
short	static	super	switch	synchronized	
this	throw	throws	transient	true	try
typeof	var	void	volatile	while	with

FIGURE 4.6: JavaScript reserved words.

Variables

- Untyped!
- Can be declared with var keyword:

```
var foo;
```

- Can be created automatically by assigning a value:

```
foo=1;    blah="Hi Dave";
```

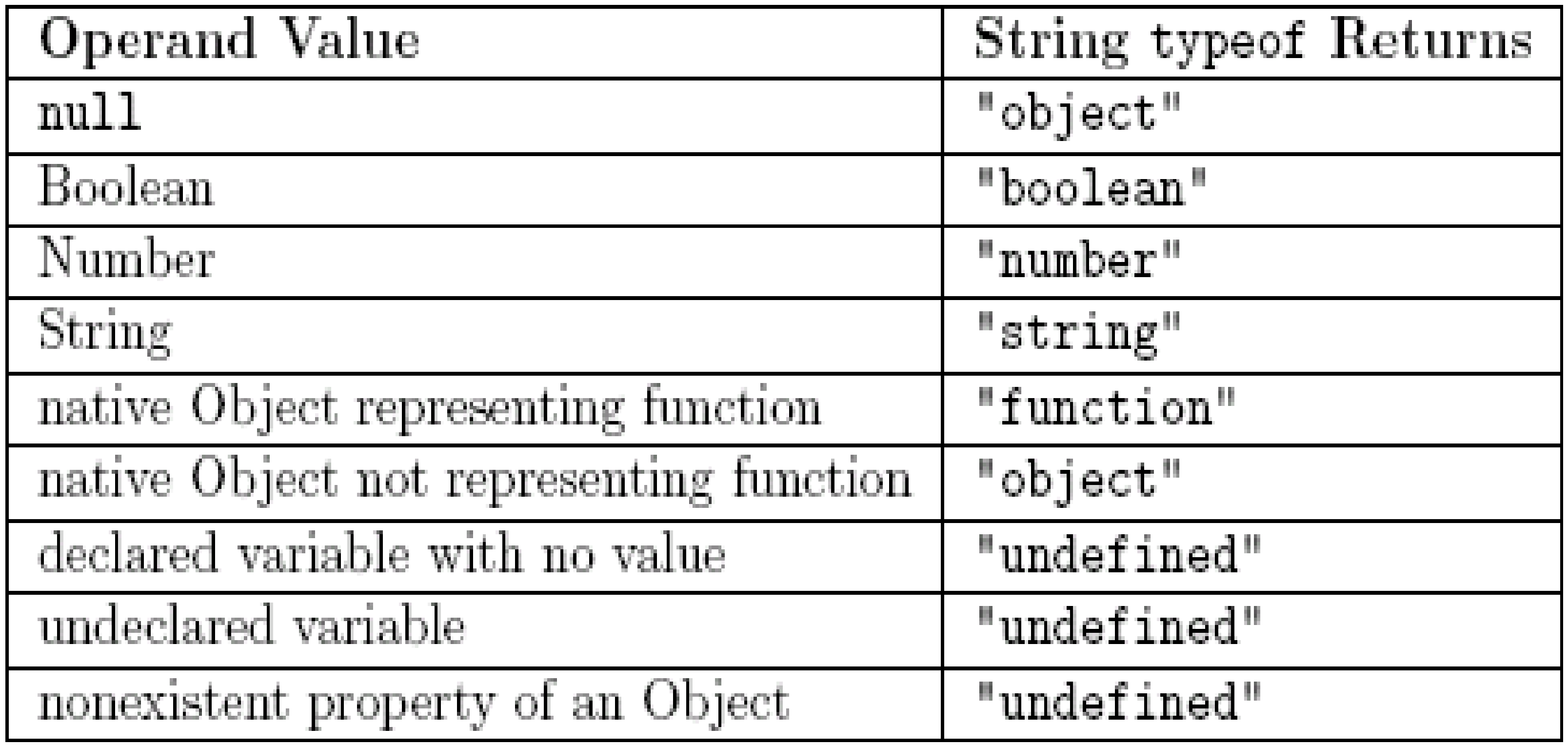
Variables

- Using **`var`** to declare a variable results in a *local* variable (inside a function).

- If you don't use **`var`** – the variable is a global variable.

Variables

TABLE 4.1: Values returned by `typeof` for various operands.



Operand Value	**String typeof Returns**
`null`	`"object"`
Boolean	`"boolean"`
Number	`"number"`
String	`"string"`
native Object representing function	`"function"`
native Object not representing function	`"object"`
declared variable with no value	`"undefined"`
undeclared variable	`"undefined"`
nonexistent property of an Object	`"undefined"`

Operators

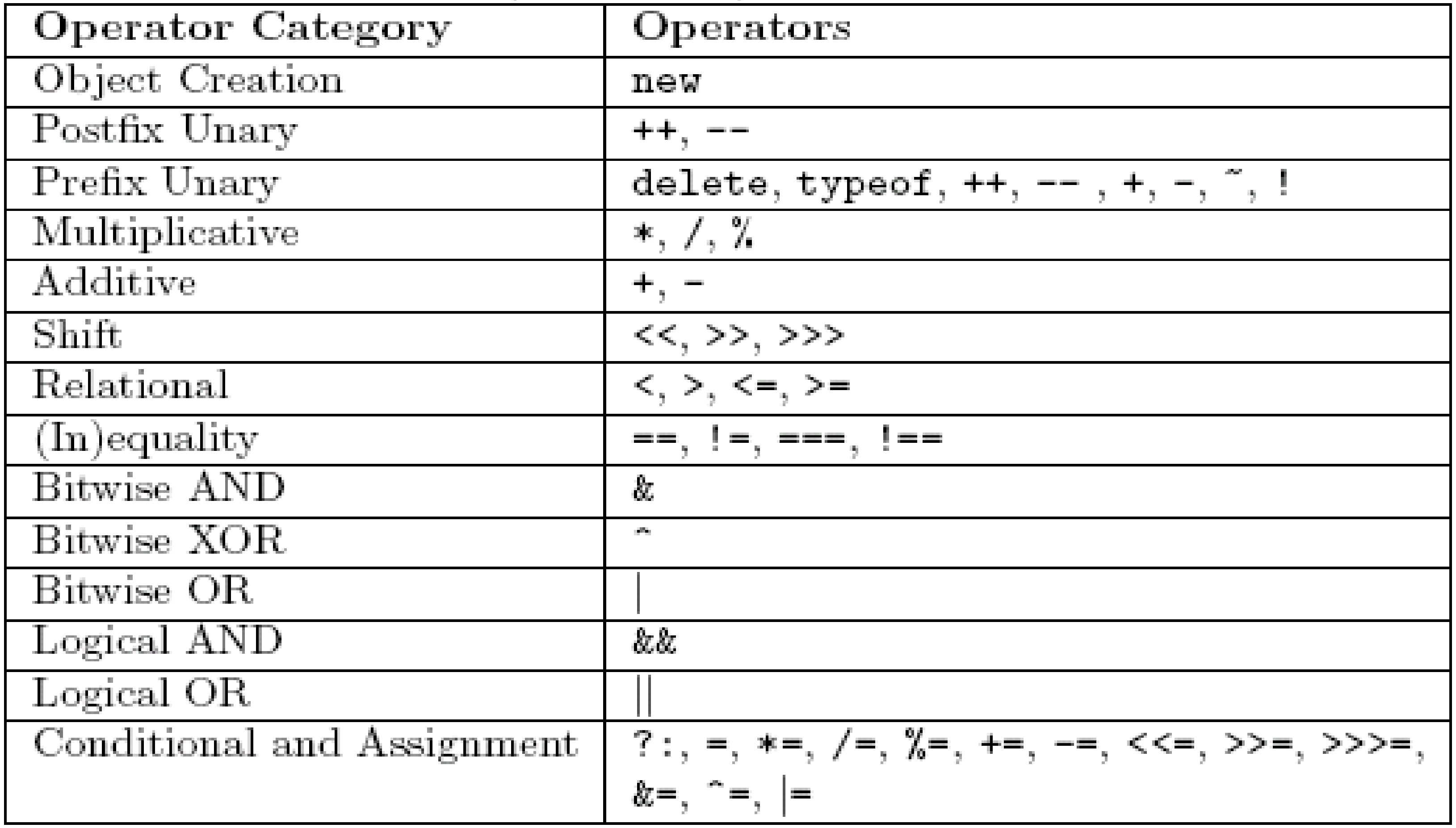


TABLE 4.6: Precedence (high to low) for selected JavaScript operators.

Operator Category	Operators
Object Creation	`new`
Postfix Unary	`++`, `--`
Prefix Unary	`delete`, `typeof`, `++`, `--` , `+`, `-`, `~`, `!`
Multiplicative	`*`, `/`, `%`
Additive	`+`, `-`
Shift	`<<`, `>>`, `>>>`
Relational	`<`, `>`, `<=`, `>=`
(In)equality	`==`, `!=`, `===`, `!==`
Bitwise AND	`&`
Bitwise XOR	`^`
Bitwise OR	`|`
Logical AND	`&&`
Logical OR	`||`
Conditional and Assignment	`?:`, `=`, `*=`, `/=`, `%=`, `+=`, `-=`, `<<=`, `>>=`, `>>>=`, `&=`, `^=`, `|=`

JavaScript Operators

- Associativity:
  - Assignment, conditional, and prefix unary operators are right associative: equal-precedence operators are evaluated right-to-left:

`a *= b += c` $\Longleftrightarrow$ `a *= (b += c)`

  - Other operators are left associative: equal-precedence operators are evaluated left-to-right

JavaScript Operators: Automatic Type Conversion

- Binary operators +, −, *, /, % convert both operands to Number
  - Exception: If one of operands of + is String then the other is converted to String
- Relational operators <, >, <=, >= convert both operands to Number
  - Exception: If both operands are String, no conversion is performed and lexicographic string comparison is performed

JavaScript Operators: Automatic Type Conversion

- Operators ===, !== are strict:
  - Two operands are === only if they are of the same type and have the same value
  - "Same value" for objects means that the operands are references to the same object
- Unary +, - convert their operand to Number
- Logical &&, ||, ! convert their operands to Boolean (normally)

JavaScript Operators

- Bit operators
  - Same set as Java:
    - Bitwise NOT, AND, OR, XOR (~, &, |, ^)
    - Shift operators (<<, >>, >>>)
  - Semantics:
    - Operands converted to Number, truncated to integer if float

JavaScript Statements

- Expression statement: any statement that consists entirely of an expression
  - Expression: code that represents a value

```
i = 5;
j++;
```

- Block statement: one or more statements enclosed in { } braces
- Keyword statement: statement beginning with a keyword, *e.g.*, `var` or `if`

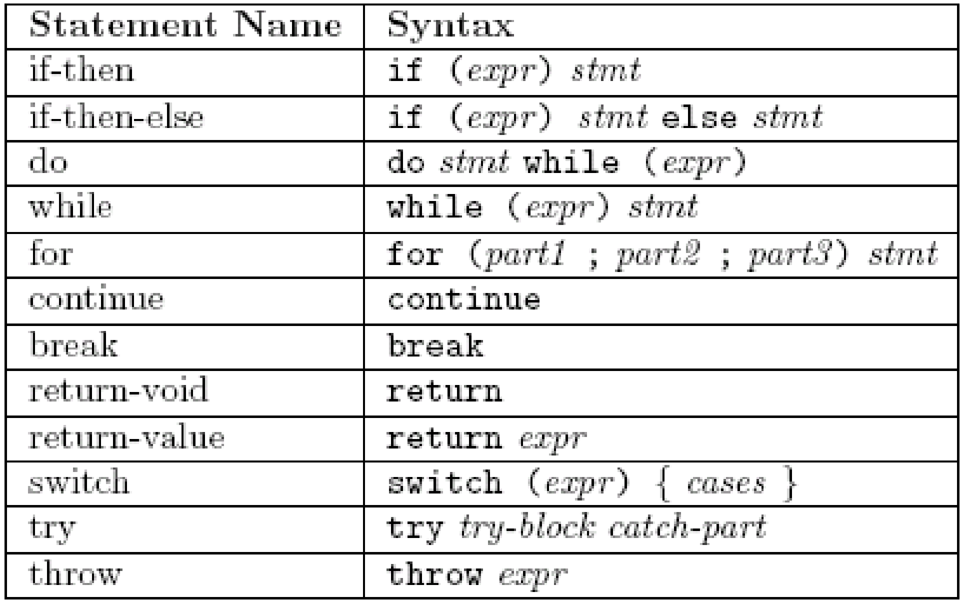
JavaScript Statements

- `var` syntax: `var i, msg="hi", o=null;`

  Comma-separated declaration list with optional initializers

- Java-like keyword statements:

TABLE 4.5: JavaScript keyword statements.



Statement Name	Syntax
if-then	`if (`*expr*`)` *stmt*
if-then-else	`if (`*expr*`)` *stmt* `else` *stmt*
do	`do` *stmt* `while (`*expr*`)`
while	`while (`*expr*`)` *stmt*
for	`for (`*part1* `;` *part2* `;` *part3*`)` *stmt*
continue	`continue`
break	`break`
return-void	`return`
return-value	`return` *expr*
switch	`switch (`*expr*`) {` *cases* `}`
try	`try` *try-block catch-part*
throw	`throw` *expr*

Basic JavaScript Syntax

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      "  What is it?", "");
```

Basic JavaScript Syntax

Notice that there is no main() function/method

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      "  What is it?", "");
```

Basic JavaScript Syntax

Comments like Java/C++ (/* */ also allowed)

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      "  What is it?", "");
```

Basic JavaScript Syntax

Variable declarations:
- Not required
- Data type not specified

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      "  What is it?", "");
```

Basic JavaScript Syntax

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      " What is it?", "");
```

Semi-colons are usually not required, but always allowed at statement end

Basic JavaScript Syntax

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      "  What is it?", "");
```

Arithmetic operators same as Java/C++

Basic JavaScript Syntax

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      "  What is it?", "");
```

String concatenation operator as well as addition

Basic JavaScript Syntax

```
// HighLow.js

var thinkingOf;  // Number the computer has chosen (1 through 1000)
var guess;       // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      " What is it?", "");
```

Arguments can be any expressions

Argument lists are comma-separated

Basic JavaScript Syntax

```
// HighLow.js

var thinkingOf;   // Number the computer has chosen (1 through 1000)
var guess;        // User's latest guess

// Initialize the computer's number
thinkingOf = Math.ceil(Math.random()*1000);

// Play until user guesses the number
guess = window.prompt("I'm thinking of a number between 1 and 1000." +
                      "  What is it?", "");
```

Object dot notation for method calls as in Java/C++

Basic JavaScript Syntax

```
while (guess != thinkingOf)
{

  // Evaluate the user's guess
  if (guess < thinkingOf) {
     guess = window.prompt("Your guess of " + guess +
                           " was too low.  Guess again.", "");
  }
  else {
     guess = window.prompt("Your guess of " + guess +
                           " was too high.  Guess again.", "");
  }
}

// Game over; congratulate the user
window.alert(guess + " is correct!");
```

Basic JavaScript Syntax

Many control constructs and use of { } identical to Java/C++

```
while (guess != thinkingOf)
{

  // Evaluate the user's guess
  if (guess < thinkingOf) {
     guess = window.prompt("Your guess of " + guess +
                           " was too low.  Guess again.", "");
  }
  else {
     guess = window.prompt("Your guess of " + guess +
                           " was too high.  Guess again.", "");
  }
}

// Game over; congratulate the user
window.alert(guess + " is correct!");
```

Basic JavaScript Syntax

Most relational operators syntactically same as Java/C++

```
while (guess != thinkingOf)
{

  // Evaluate the user's guess
  if (guess < thinkingOf) {
     guess = window.prompt("Your guess of " + guess +
                           " was too low.  Guess again.", "");
  }
  else {
     guess = window.prompt("Your guess of " + guess +
                           " was too high.  Guess again.", "");
  }
}

// Game over; congratulate the user
window.alert(guess + " is correct!");
```

Basic JavaScript Syntax

```
while (guess != thinkingOf)
{

  // Evaluate the user's guess
  if (guess < thinkingOf) {
     guess = window.prompt("Your guess of " + guess +
                           " was too low.  Guess again.", "");
  }
  else {
     guess = window.prompt("Your guess of " + guess +
                           " was too high.  Guess again.", "");
  }
}

// Game over; congratulate the user
window.alert(guess + " is correct!");
```

Automatic type conversion:
`guess` is String,
`thinkingOf` is Number

JavaScript Functions

- Function declaration syntax

```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
```

JavaScript Functions

- Function declaration syntax

Declaration always begins with keyword `function`, no return type

```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
```

JavaScript Functions

- Function declaration syntax

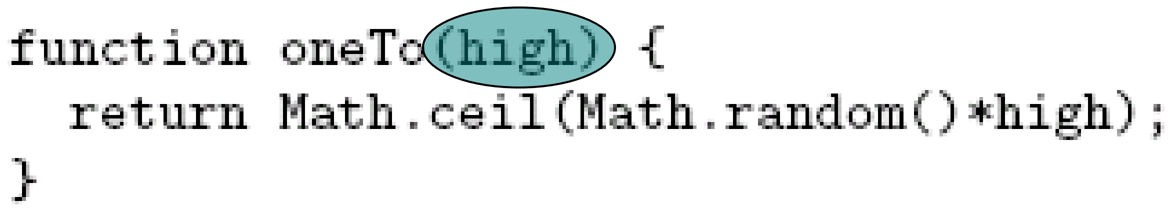
Identifier representing
function’s *name*

```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
```

JavaScript Functions

- Function declaration syntax

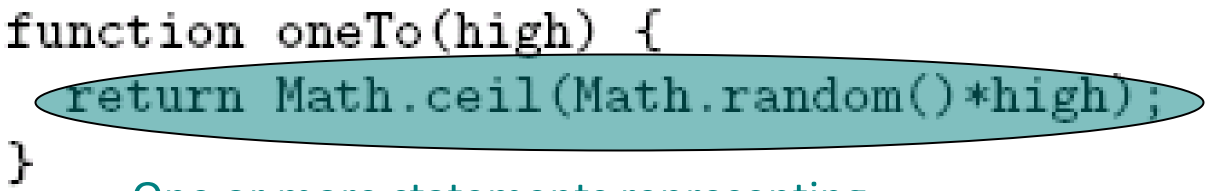
Formal parameter list



```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
```

JavaScript Functions

- Function declaration syntax



```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
```

One or more statements representing *function body*

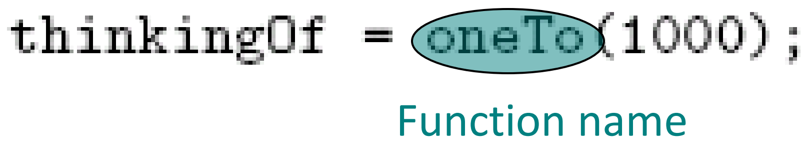
JavaScript Functions

- Function call syntax

```
thinkingOf = oneTo(1000);
```

JavaScript Functions

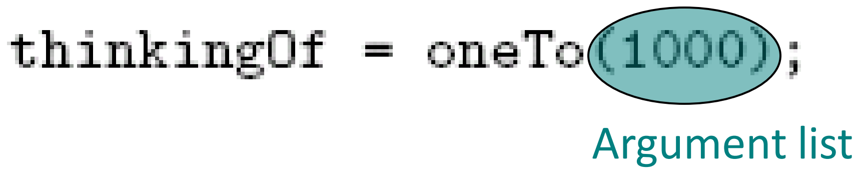
- Function call syntax



```
thinkingOf = oneTo(1000);
```

JavaScript Functions

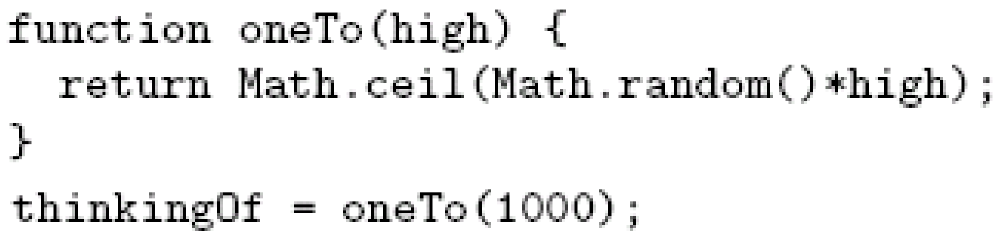
- Function call syntax



```
thinkingOf = oneTo(1000);
```

JavaScript Functions

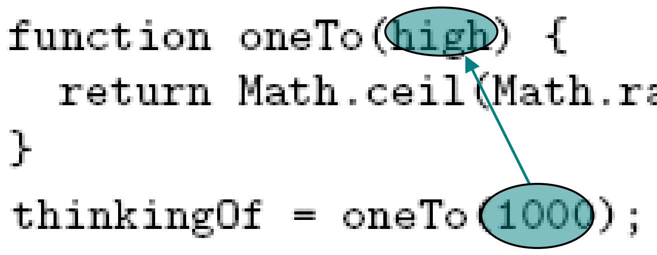
- Function call semantics:



```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
thinkingOf = oneTo(1000);
```

JavaScript Functions

- Function call semantics:



```
function oneTo(high) {
  return Math.ceil(Math.random()*high);
}
thinkingOf = oneTo(1000);
```

Argument value(s) associated with corresponding formal parameters

JavaScript Functions

- Number mismatch between argument list and formal parameter list:
  - More arguments: excess ignored
  - Fewer arguments: remaining parameters are Undefined