JavaScript Introduction

Topics discussed this presentation

- Identifiers
- Comments
- Reserved Words
- Operators
- Control Flow
- Truthy and Falsy

Identifiers

Style Guide Requirements

- May comprise only letters, numbers, $ sign, underscore
- Avoid single letter names.
- camelCase for objects, functions, instances.
- PascalCase for constructors, classes.
- Leading underscore _ for private properties.
- Number disallowed as first character.

```
function q() {...} // bad
function query {...} // good: descriptive
const my_object = {} // bad
const myObject = {} // good: camelCase
const good = new User({...}); // good: PascalCase
const $domElement = $(this).get(0); // good: jQuery variable
```

Reserved words

Java Overlap

- `break`
- `case`
- `catch`
- `continue`
- `debugger`
- `default`
- `delete`
- `do`
- `else`
- `finally`
- `for`
- `function`
- `if`
- `in`
- `instanceof`
- `new`
- `return`
- `switch`
- `this`
- `throw`
- `try`
- `typeof`
- `var`
- `void`
- `while`
- `with`

- Significant overlap with Java
- However, meaning often different in subtle ways

Comments

Single line comments

- Use / / for single line comment.
  - Position on new line above target of comment.
  - If not start block, add blank line before comment.

```
function getRadius(){
  / /  return radius of circle.
  ...
 }
```

Comments

Multi-line comments

- Use /** ... */ for multi-line comments.
  - Include description.
  - Specify parameter types and values.
  - Specify return type and value.

```
/*
 * find() returns sought value based on parameter key.
 *
 * @param {String} key
 * @return {Value} value.
 */
function find(key){
  // ...
  return value;
}
```

Operators

Assignment

```
const x = 5;
const y = 2;
let z = x + y;
z *= 2; // => 18
z /= 3; // => 6
x % 3; // => 2
```

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Operators

Arithmetic

```
const x = 5;
const y = 2;
const z = x * y; // => 10
z--; // => 9
```

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

Operators

String

```
let txt1 = 'What a very ';
txt1 += 'nice day.';
```

```
// Output
What a very nice day.
```

Operators

Add Number to String

```
const x = 5 + 5;          // => 10
const y = '5' + 5;        // => 55
const z = 'Hello' + 5;    // => Hello5
```

Operators

Comparison & Logical

```
const s = '5';
const n = 5;
s == n // => true (coercion)
s === n // => false (strict)
```

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Operators

Type

```
const car = 'Nissan';
typeof car; // => string

const cars = ['Saab', 'Volvo', 'BMW'];
cars instanceof Array; // => true
```

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Operators

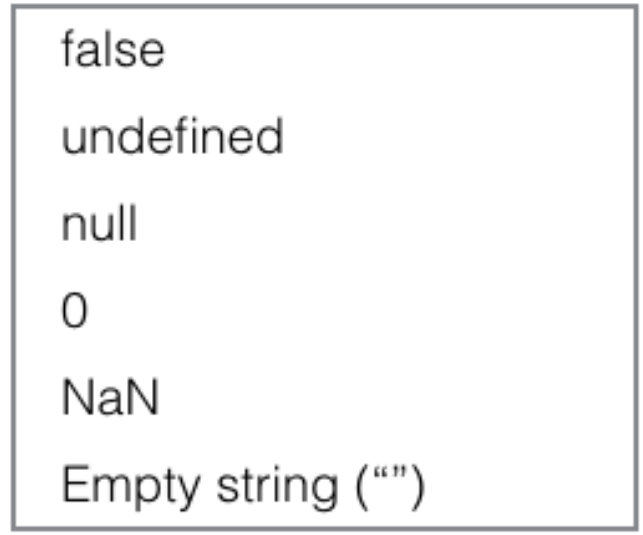
Equals and not equals

```
// Use always
===
!==
// Evil twin (Crockford)
==
!=
```

Control Flow

Truthy and Falsy

- Expression either truthy or falsy.
- Some developers avoid use.
- Not reserved words.



Falsy: these evaluate to false

Control Flow

Truthy and Falsy

Falsy

- false
- null
- undefined
- Empty string ”
- Number 0
- NaN

Truthy

- All values not truthy
- Warning: string ’false’ is truthy

Control Flow

Truthy and Falsy

Function to determine if value falsy

```
/**
 * Determines if argument resolves to a falsy or truthy value.
 *
 * @see https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
 * @param arg Argument to be checked if falsy.
 * @return Returns true if argument is falsy, otherwise false.
 */
 function falsy(arg){
   return [false, null, undefined, '', 0, NaN].includes(arg);
 }
```

Control Flow

Logical Operators (AND, OR)

Significantly different behaviour to Java

```
x && y // => x if x falsy otherwise y
x || y // => x if x truthy otherwise y

// Use OR to insert default values
const person = {};
let name = person.name || 'No such person';
console.log(name); // => No such person

person.name = 'Jane Doe';
name = person.name || 'no such person';
console.log(name); // => Jane Doe
```

Control Flow

Logical Operators (AND, OR)

Significantly different behaviour to Java

```
x && y // => x if x falsy otherwise y
x || y // => x if x truthy otherwise y

// Use AND to avoid TypeError exception
const flight = {};
flight.airline; // => undefined
flight.airline.nationality; // => TypeError
flight.airline && flight.airline.nationality; // undefined
```

Control Flow

Loop using **for**

```
const limit = 5;
for (let i = 0; i < limit; i++) {
   text += 'The number is ' + i + '<br>';
}
```

```
for (i = 0; i < 5; i++) {
  text += 'The number is ' + i + '<br>';
}
```

Control Flow

Using **while**

```
let i = 0;
while (i < 10) {
  text += 'The number is ' + i;
  i++;
}
```

```
while (condition) {
    code block to be executed
}
```

Control Flow

Using **do-while**

```
let i = 0;
const limit = 10;
do {
  text += 'The number is ' + i;
  i++;
} while (i < limit);
```

```
do {
    code block to be executed
}
while (condition);
```

Control Flow

Using **if-else**

```
if (hour < 18) {
    greeting = 'Good day';
} else{
    greeting = 'Good evening';
}
```

```
if (condition) {
    block of code to be executed if the condition is true
} else {
    block of code to be executed if the condition is false
}
```

Control Flow

Using **break** statement

break statement

- Checks for condition
- If met then immediately exits loop

```
// This trivial function return 0
function foo()
{
  const limit = 10;
  const sum = 0;
  for (let i = 0; i < limit; i += 1) {
    if (i % 2 == 0)
      break;
    else
      sum += i;
  }

  return sum;
};
```

Control Flow

Using **continue** statement

continue statement

- Checks for condition
- If met, skips remainder iteration
- Continues with next iteration

```
// Calculate sum of odd integers in range [0, 10]
function addOdd(){
  const limit = 10;
  let sum = 0;
  for (let i = 0; i < limit; i += 1) {
    if (i % 2 == 0)
      continue;
    else
      sum += i;
  }

  return sum;
};
```

Control Flow

Ternary (Conditional) Operator

```
// If argument a is negative return -a.
      Otherwise, return a
function absoluteValue(a)
{
  return a < 0 ? -a : a;
};
```

```
variablename = (condition) ? value1:value2
```

Control Flow

Logical Operators

```
function foo(){
  const limit = 5;
  let i = 0;
  let j = 0;
  let sum = 0;
  while (++i < limit && ++j < limit){
    sum += i + j;
  }

  return sum;
};

console.log(foo());  // 20
```

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

Control Flow

Using **switch** statement

```
switch (new Date().getDay()) {
    case 0:
    case 6:
        day =  'Weekend';
        break;
    default:
        day  = 'Weekday';
}
```

```
switch(expression) {
    case n:
        code block
        break;
    case n:
        code block
        break;
    default:
        default code block
}
```