Type checks

```
console.log(typeof ""); // "string"
console.log(typeof "hello"); // "string"
console.log(typeof String("hello")); // "string"
console.log(typeof new String("hello")); // "object"

console.log(typeof 0); // "number"
console.log(typeof -0); // "number"
console.log(typeof 0xff); // "number"
console.log(typeof -3.142); // "number"
console.log(typeof Infinity); // "number"
console.log(typeof -Infinity); // "number"
console.log(typeof NaN); // "number"
console.log(typeof Number(53)); // "number"
console.log(typeof new Number(53)); // "object"

console.log(typeof true); // "boolean"
console.log(typeof false); // "boolean"
console.log(typeof new Boolean(true)); // "object"

console.log(typeof undefined); // "undefined"

console.log(typeof null); // "object"

console.log(typeof Symbol()); // "symbol"

console.log(typeof []); // "object"
console.log(typeof Array(5)); // "object"

console.log(typeof function() {}); // "function"
console.log(typeof new Function); // "function"

console.log(typeof new Date); // "object"

console.log(typeof /^(.+)$/); // "object"
console.log(typeof new RegExp("^(.+)$")); // "object"

console.log(typeof {}); // "object"
console.log(typeof new Object); // "object"
```

Checking for null

```
function isNull(value) {
    return value === null;
}
console.log(undefined == null); // true
console.log(undefined === null); // falsev
```

Checking for NaN

```
console.log(isNaN(NaN)); // true
console.log(isNaN(null)); // false
console.log(isNaN(undefined)); // true
console.log(isNaN(Infinity)); // false

console.log(Number.isNaN(NaN)); // true
console.log(Number.isNaN(null)); // false
console.log(Number.isNaN(undefined)); // false
console.log(Number.isNaN(Infinity)); // false

var x = NaN;

console.log(x == NaN); // false
console.log(x === NaN); // false

function isNan(value) {
  return value !== value;
}

Number.isNaN = Number.isNaN || (function(value) {
  return value !== value;
})

function isNan(value) {
  return Object.is(value, Number.NaN);
}
```

Checking for arrays

```
// METHOD 1: constructor property
// Not reliable
function isArray(value) {
  return typeof value == 'object' && value.constructor
   === Array;
}

// METHOD 2: instanceof
// Not reliable since an object's prototype can be
   changed
// Unexpected results within frames
function isArray(value) {
  return value instanceof Array;
}

// METHOD 3: Object.prototype.toString()
// Better option and very similar to ES6 Array.isArray()
function isArray(value) {
  return Object.prototype.toString.call(value) ===
'[object Array]';
}

// METHOD 4: ES6 Array.isArray()
function isArray(value) {
  return Array.isArray(value);
}
```

Generic type checking

```
console.log(type('')); // "string"
console.log(type('hello')); // "string"
console.log(type(String('hello'))); // "string"
console.log(type(new String('hello'))); // "string"

console.log(type(0)); // "number"
console.log(type(-0)); // "number"
console.log(type(0xff)); // "number"
console.log(type(-3.142)); // "number"
console.log(type(Infinity)); // "number"
console.log(type(-Infinity)); // "number"
console.log(type(NaN)); // "number"
console.log(type(Number(53))); // "number"
console.log(type(new Number(53))); // "number"

console.log(type(true)); // "boolean"
console.log(type(false)); // "boolean"
console.log(type(new Boolean(true))); // "boolean"

console.log(type(undefined)); // "undefined"

console.log(type(null)); // "null"

console.log(type(Symbol())); // "symbol"
console.log(type(Symbol.species)); // "symbol"

console.log(type([])); // "array"
console.log(type(Array(5))); // "array"

console.log((function() { return type(arguments) })());
// "arguments"
console.log(type(function() {})); // "function"
console.log(type(new Function)); // "function"

console.log(type(class {})); // "function"

console.log(type({})); // "object"
console.log(type(new Object)); // "object"
```