JavaScript Introduction

Topics discussed

- Functions
- Anonymous functions
- Arrow function
- Spread/Rest operator/parameters
- **this** binding
- Anonymous function as function parameter

JavaScript

Functions

function

- Block of code defined once
- Invokable many times
- May include parameters
- Observe differences Java
- Functions attached to objects referred to as *methods*
- Functions are objects
  - Assignable to variable
  - Allowable as parameter

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

console.log(square(10)); // => 100
```

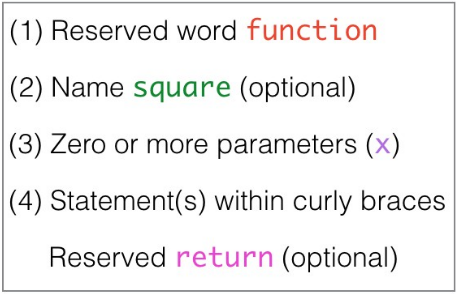
```
function add() {
  let counter = 0;
  function plus() {counter += 1;}

  plus();
  return counter;
}

console.log(add()); // => 1
```

JavaScript

Function has four parts



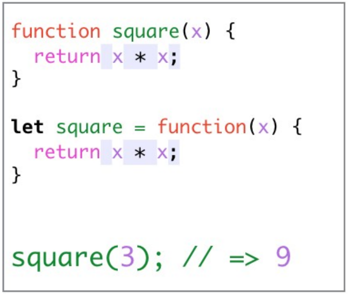
(1) Reserved word `function`

(2) Name `square` (optional)

(3) Zero or more parameters (`x`)

(4) Statement(s) within curly braces

Reserved `return` (optional)



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

let square = function(x) {
  return x * x;
}


square(3); // => 9
```

Function

Hidden parameters

Every function has 2 hidden parameters

- **this**
  - Reference determined by which of four available function invocation patterns used.
- **arguments**
  - Array type object containing all parameters.
  - Treat as obsolete, instead use rest arguments.
  - Rest arguments a real Array, not Array-like like arguments

```
let anObject = {
  value: 0,
  increment: function () {
    this.value += 1;
  },
};

// Output: 1
anObject.increment();
```

```
function aFunction(...args){
  return args.length;
}

// Output: 2
console.log(aFunction(3, 4));
```

Function

Rules re function arguments

- Function declaration states arguments (parameters).
- Pass too many, extra args ignored.
- Pass too few, missing args assigned *undefined*.
- Here we use deprecated *arguments* object.

```
function foo(x){
  console.log('x: ',x, 'arguments: ',arguments);
}

foo(); // x: undefined arguments: [] (missing arg)
foo(10); // x: 10 arguments [10] (correct number args)
foo(10, "arg 2"); // x: 10 arguments: [10, "arg 2"] (extra arg)
```

Function

Spread/Rest operator (ES6)

arguments now deprecated - use Spread/Rest

- operator comprises three periods (. . .)
- This example . . . is **rest** operator
- Alternatively in MDN: **rest** parameters

```
/**
 * Example: Function defined to take variable number   arguments
 * @see MDN rest parameters (operator)
 */
function foo(...args) {
console.log(args);
}

foo( 1, 2, 3, 4, 5); //   [1,2,3,4,5]
```

Function

Spread/Rest operator (ES6)

Example: the power of spread/rest operator

- This example . . . is **spread** operator

```
/**
 * Example: assembling new array
 * @see MDN Spread operator
 */
let parts = ['shoulders', 'knees'];
let all = ['head', ...parts, 'and', 'toes'];
console.log(all); //['head', 'shoulders', 'knees', 'and', 'toes'];
```

Function

Spread/Rest operator (ES6)

```
//Deprecated arguments hidden  parameter
function add (x, y) {
  console.log(arguments);
  return x + y;
}

add(1, 2); //  [1, 2]
```

```
/**
 * ES6  Using Spread|Rest operator
 * Determine number parameters at  runtime
 * @see ES6 &  Beyond page 13  (referenced)
 */
function multiply(...args){
  console.log(args);
}

multiply(3,4); //  [3, 4]
```

Functions

Invocation Patterns

Four function invocation patterns:

- 1. Method invocation
  - **this** bound to containing object
  - function is method - a property of containing object
- 2. Function invocation
  - **this** bound to global object
  - function property of global object
- 3. Constructor invocation
  - **this** bound to containing object
  - **new** not used: this bound to global
- 4. Apply invocation
  - Outside course scope

```
let anObject = {
  value: 0,
  increment: function () {
    this.value += 1;
  },
};

// method invocation
anObject.increment();
```

```
value = 0;
function increment(){
  this.value += 1;
};

// function invocation
increment();
```

JavaScript

this binding

Note: behaviour different in strict mode

```
// Function invocation: this bound to global object
function set(x){
  this.x= x;
  console.log(x); // => 100
};
set(100); // sets global variable x to 100
```

```
// Here, because of strict mode, this is undefined
'use strict';
function set(x){
  this.x = x; // => TypeError
  console.log(x);
};
set(100); // fails due to TypeError
```

JavaScript

this binding

```
// Method invocation: this bound to containing object
const myObj = {
  x: 100,
  set: function (x) {
    this.x = x;
    return this;
  },
};
myObj.set(100);// sets myObj.x to 100
console.log(myObj); // Object {x: 100}
console.log(myObj.set(100)); // Object {x: 100}
```

JavaScript

this binding

strict mode causes different behaviour:

- 'use strict';
- Prevents access to global variable
- **this** undefined
- TypeError generated when code below run in strict mode

```
// Method invocation: this now bound to global object
myObj = {
  x: 0,
  set: function (x) {
    modify(x);
    function modify(val) { // nested function
      this.x = x; // this bound to global obj: undefined in strict mode
    };
  },
};
```

JavaScript

=> arrow function

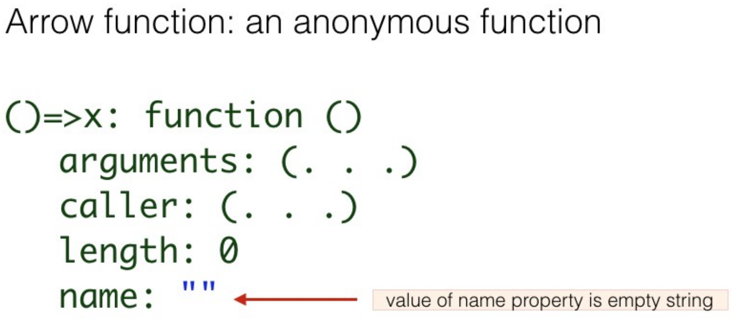
```
// What we're familiar with:
function add(x, y) {
  return x + y;
}

console.log(add(10, 20)); // 30
```

```
/**
 * Alternative approach: arrow function.
 * @see page 46 ES6 and Beyond (referenced)
 * @see MDN (referenced)
 */
const add2 = (x, y) => x + y;
console.log(add2(10, 20)); // 30
```

JavaScript

anonymous function: value of its property *name* is empty string



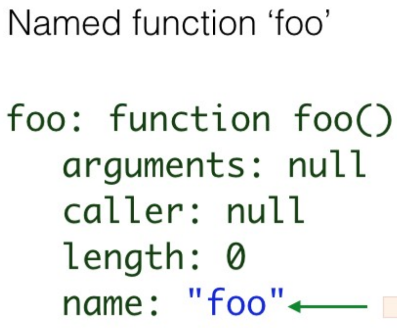
Arrow function: an anonymous function

```
()=>x: function ()
   arguments: (. . .)
   caller: (. . .)
   length: 0
   name: ""
```

value of name property is empty string

JavaScript

named function



Named function ‘foo’

```
foo: function foo()
   arguments: null
   caller: null
   length: 0
   name: "foo"
```

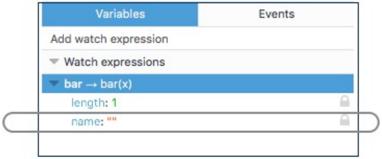
value of name property is string “foo”

JavaScript

anonymous function

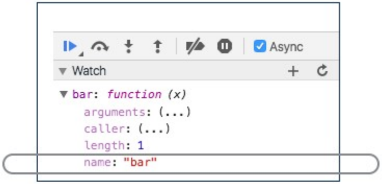
- The code snippet illustrated is legal.
- But note different browser treatments.

```
const bar = (x)=>{console.log(x);}
bar(10);
```



```
Variables    Events
Add watch expression
▼ Watch expressions
▼ bar → bar(x)
    length: 1
    name: ""
```

Firefox



```
▼ Watch
▼ bar: function (x)
    arguments: (...)
    caller: (...)
    length: 1
    name: "bar"
```

Chrome

JavaScript

this binding

Pre-ES6 workaround hack

```
'use strict';
let myObj = {
  x: 0,
  set: function (x) {
    let that = this;
    modify(x);
    function modify(val) { // nested function
      that.x = x; // workaround hack
    };
  },
};

myObj.set(100); // myObj.x set to 100
```

JavaScript

this binding

Use arrow function to bind inner **this** to containing object

```
//this now bound to containing object myObj
'use strict';
let myObj = {
  x: 0,
  set: function (x) {
    let modify = (val) => { // nested function
      this.x = val;// this now bound to myObj
      console.log(this); // Object{x: 0}
    };

    modify(x);
  },
};

console.log(myObj); // Object { x: 100, set: myObj.set() }
myObj.set(100); // myObj.x set to 100
console.log(myObj.x); // 100
```

JavaScript

this binding

Another JavaScript booby trap

```
// Okay: Method invocation: this bound to containing object
myObj = {
  x: 0,
  set: function (x) {
    this.x = x;
    return this;
  },
};

console.log(myObj); // Object {x: 0}
console.log(myObj.set(0)); // Object {x: 0}
```

JavaScript

this binding

Another JavaScript booby trap

```
/**
 * Not okay: Alternative approach: arrow function.
 * Method invocation: this now bound to global object
 * Here arrow function is NOT an inner function.
 * As inner function its this would be bound to containing function.
 * @see page 50 ES6 and Beyond (referenced)
 */
myObj = {
  x: 0,
  set: x => {
    this.x = x;
    return this;
  },
};

console.log(myObj); // Object {x: 0}
console.log(myObj.set(0)); // Window {...}
```

JavaScript

this binding

Constructor invocation: not recommended

```
'use strict';
function Person(name){
   this.name = name; // this bound to Person object
}

let x = new Person('Jane');
console.log(x); // Object { name: "Jane"}
```

```
// Omitting 'use strict'
// If strict mode & new omitted then this undefined
function Person(name){
   this.name = name; // this bound to global object
}

let x = Person('Jane'); // Oops! Forgot new keyword
console.log(x); // undefined
```

JavaScript

Passing function as function argument

```
// Passing a named function as an argument
function myFn(fn){
  const result = fn();
  console.log(result);
};

function myOtherFn(){
  return 'hello world';
};

// logs 'hello world'
myFn(myOtherFn);
```

Functions

Which to use? Function expression or function statement

```
// Function statements: Airbnb recommendation (ES6)
function outer1(){
  hoisted(); // => foo
  function hoisted(){
    console.log('foo');
  }
}
```

```
// Function expressions: Crockford recommendation (ES5)
let outer2 = function outer2() {
  notHoisted(); // => TypeError: notHoisted is not a function
  let notHoisted = function() {
     console.log('bar');
  };
};
```

Functions

Example use arrow function as function parameter

First attempt: method to filter even numbers.

```
'use strict';

const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

function filter(numbers) {
  const filterNumbers = [];
  let j = 0;
  for (let i = 0; i < numbers.length; i += 1) {
    if (numbers[i] % 2 === 0) {
      filterNumbers[j] = numbers[i];
      j += 1;
    }
  }
  return filterNumbers;
}
console.log(filter(array)); // [2, 4, 6, 8, 10]
```

Functions

Example use arrow function as function parameter

Second attempt: use bespoke function & Array.filter

```
'use strict';

const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

function even(x) {
  return x % 2 === 0;
}

console.log(array.filter(even)); // [2, 4, 6, 8, 10]
```

Functions

Example use arrow function as function parameter

Production: Use arrow function and Array.filter method

```
'use strict';

const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

console.log(array.filter(x => x % 2 === 0)); // [2, 4, 6, 8, 10]
```

Functions

Example use arrow function as function parameter

About 80% reduction in code size

```
'use strict';
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
function filter(numbers) {
  const filterNumbers = [];
  let j = 0;
  for (let i = 0; i < numbers.length; i += 1)  {
    if (numbers[i] % 2 === 0) {
      filterNumbers[j] = numbers[i];
      j += 1;
    }
  }
  return filterNumbers;
}
console.log(filter(array)); //[2, 4, 6, 8, 10]
```

```
'use strict';
const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
console.log(array.filter(x => x % 2 === 0));
```

Summary

- A JavaScript function:
    - Is a first class object,
        - Like any other object
    - Assignable to a variable,
    - May be anonymous,
    - Legal as parameter in function call,
    - May be a value in an object,
    - Could contain other functions,
    - Arrow function (ES6) form available.
- Spread operator
- Rest parameters