Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

Introducción a JavaScript II

IIC2513 - Tecnologías y Aplicaciones Web

Sebastián Vicencio R.
2do Semestre 2020

Funciones

Encapsular código

Funciones

Valor no primitivo

- Encapsular trozo código en un valor
- Evitar repetir código
- Se “llama” para ejecutar código

```
function fnName(param1, param2, ..., paramN) {
  // Statements (body)
}
```

Funciones

Ejecución

```
fnName('This is a parameter')
```

La llamada es una expresión

```
return expression;
```

Valor de retorno

Funciones

Como un valor

```
function (str) {
  console.log(“Logging", str);
}
```

```
let myLogger = function (str) {
  console.log(“Logging", str);
}
```

Se puede llamar una vez asignada a una variable

Funciones

Como un valor

```
let writeCourseInfo = function(writer) {
  writer('IIC2513 – Tecnologías y Aplicaciones Web');
}

writeCourseInfo(myLogger);
```

Se trata como cualquier otro valor

Scope

Visibilidad de variables

Scope

Generación de scope local

Scope de función

Toda función genera un nuevo scope

Variables fuera de función no son visibles

```
function diffScope() {
  var a = 5;
  let b = 'times';
  console.log(a, b);
}

diffScope();

console.log(a);
// ReferenceError: a is not defined
console.log(b);
// ReferenceError: b is not defined
```

Scope

Generación de scope local

Scope de bloque

(Casi) todo bloque genera un nuevo scope

Con let y const sí

Con var no

```
if (true) {
  let a = 5;
}

console.log(a);
// ReferenceError: a is not defined
```

Scope

Generación de scope local

Scope de bloque

(Casi) todo bloque genera un nuevo scope

Con let y const sí
Con var no

```
if (false) {
  var a = 5;
}

console.log(a); // undefined
// Hoisted

if (true) {
  var a = 5;
}

console.log(a); // 5
```

Arrow functions

Funciones abreviadas

```
let arrowFn = (param) => {
  return !!param;
}

let arrowFnReturn = (param) => !!param;
```

Optional arguments

¿Qué pasa si no pasamos los argumentos exactos?

- Argumentos extra: ignorados
- Argumentos faltantes: **`undefined`**
- Argumentos opcionales

```
let defaultFn = function(param = 'defaultValue') {
  return param;
}
defaultFn();
```

Closure

Scopes en juego

Closure

Función que referencia variables del scope local que la rodea

```
let sayLoud = function(param) {
  const imperativeParam = param + ‘!’;

  return () => console.log(imperativeParam);
}

const sayHi = sayLoud('Hi');
sayHi();
// Hi!
```

Se suele usar en una función dentro de otra función

Closure

Un ejemplo más real

```
function multiplier(factor) {
  return number => number * factor;
}

let twice = multiplier(2);
let fiveTimes = multiplier(5);
console.log(twice(5));
console.log(fiveTimes(11));
```

Pure functions

```
function add(a, b) {
  return a + b;
}
```

Función pura

Mismos argumentos, mismo retorno

No tiene side effects

```
function myLogger(str) {
  console.log("Logging", str);
}
```

Función no pura

Imprimir es un side effect

Arrays

Listas de valores

Estructura con múltiples valores

Arrays

Creación

```
let newArray = []; // Empty

let randomArray = [1, 'text', 10];
```

Arrays

Acceso - Elementos indexados

```
let randomArray = [1, 'text', 10];
const index = 0;


randomArray[index]; // 1
randomArray[1]; // 'text'


randomArray[1] = 'Yes';
```

Arrays

Métodos relevantes

Son objetos

Objetos tienen propiedades y métodos

```
randomArray.push(true);

for (let i = 0; i < randomArray.length; i += 1) {
  console.log(randomArray[i]);
}
```

Arrays

Métodos relevantes

- `pop`: remueve último elemento
- `join`: junta elementos en string
- `shift`: remueve primer elemento
- `unshift`: agregar primera posición
- `concat`: merge dos arreglos
- `forEach`: iterar
- `map`: convertir valores

```
randomArray.forEach(elem => console.log(elem));

randomArray.map(elem => elem + 30);
```

Objetos

Estructura que **agrupa datos** relacionados

Cada dato es una **propiedad**

Objetos

Propiedades

```
{
  propertyName: propertyValue
}
```

propertyName: string
propertyValue: cualquier valor

Método: propiedad cuyo valor es una función

Object literal

Inicialización

```
let myObject = {};

let person = {
  name: 'Sharon',
  age: 27
};
```

Object literal

Notación dot y square brackets

Dot

```
person.name;
person.age;
```

Square brackets

```
person['name'];
person['job']; // undefined
```

Object literal

Modificación

Misma notación, pero con asignación

```
person.age = 28; // Existing property

person.city = 'Santiago'; // New property

person['isStudent'] = false;
```

Object literal

Uso de this

Aquí `this` hace referencia al objeto mismo

```
person.talk = function() {
  console.log('My name is ' + this.name + '. I live in ' + this.city);
}
```

Constructores

¿Y si quisiéramos crear varios objetos con la misma estructura?

Constructores

Funciones “especiales”

```
function Person(name, age, city, isStudent = false) {
  this.name = name;
  this.age = age;
  this.city = city;
  this.isStudent = isStudent;
  this.talk = function() {
    console.log('My name is ' + this.name + '. I live in ' + this.city);
  }
}
```

Constructores

Instanciar objetos

```
let local = new Person('Sharon', 27, 'Santiago');

let foreigner = new Person('John', 19, 'Berlin', true);
```

Mutabilidad

Objetos pueden ser modificados

Mutabilidad

Modificar arreglo

```
let randomArray = ['one element'];

function doSomething(arr) {
  arr.push('what happens with this?');
}

doSomething(randomArray);
console.log(randomArray);
```

Mutabilidad

Modificar objeto

```
let randomObj = { one: 'element' };

function doSomethingElse(obj) {
  obj.what = 'happens with this?';
}

doSomethingElse(randomObj);
console.log(randomObj);
```

Objects everywhere

(Casi) todo es un objeto

(Casi) todo es un objeto

Incluidos valores primitivos

Un objeto es instanciado para cada valor primitivo

```
"hola".length;

true.toString();
```

Únicas excepciones: `null` - `undefined`

Actividad

Hands-on