Funcții de Primă Clasă în JavaScript

Curs 2

Gabriel Danciu

D.E.C.

27 februarie 2024

Prezentare Generală a Cursului

1. Funcții de primă clasă
2. Paradigme multiple
3. Programare imperativă
4. Programare funcțională
5. Programare bazată pe prototipare OOP
6. Metaprogramare
7. Aplicatii
8. Teme

Funcții de primă clasa

- Termenul "first-class" (de primă clasă) înseamnă că ceva este pur și simplu o valoare. O funcție de primă clasă este una care poate fi folosită oriunde poate merge orice altă valoare - există puține sau zero restricții. Un număr în JavaScript este cu siguranță un lucru de primă clasă, și prin urmare, o funcție de primă clasă are o natură similară.

Exemplu de first-class

```
var fortytwo = function() { return 42 };
var fortytwos = [42, function() { return 42 }];
var fortytwos = {number: 42, fun: function() { return 42
    }};
function weirdAdd(n, f) { return n + f() }
weirdAdd(42, function() { return 42 });
```

Paradigme multiple in Javascript

- Programare imperativă. Programarea bazată pe descrierea detaliată a acțiunilor
- Programare orientată-obiect bazată pe prototipuri. Programarea bazată pe obiecte prototip și instanțe ale acestora
- Metaprogramare Programarea care manipulează baza modelului de execuție al JavaScript

Exemplu de programare imperativă

```
var lyrics = [];
for (var bottles = 99; bottles > 0; bottles--) {
  lyrics.push(bottles + " bottles of beer on the wall");
  lyrics.push(bottles + " bottles of beer");
  lyrics.push("Take one down, pass it around");
  if (bottles > 1) {
    lyrics.push((bottles - 1) + " bottles of beer on the
        wall.");
  }
  else {
    lyrics.push("No more bottles of beer on the wall!");
  }
}
```

Exemplu de programare funcțională

```
function lyricSegment(n) {
  return _.chain([])
    .push(n + "␣bottles␣of␣beer␣on␣the␣wall")
    .push(n + "␣bottles␣of␣beer")
    .push("Take␣one␣down,␣pass␣it␣around")
    .tap(function(lyrics) {
    if (n > 1)
            lyrics.push((n - 1) + "␣bottles␣of␣beer␣on␣
                the␣wall.");
          else
            lyrics.push("No␣more␣bottles␣of␣beer␣on␣the␣
                wall!");
        })
    .value();
}
lyricSegment(9);
```

Programare bazată pe prototipare OOP

- Când o funcție este creată în afara contextului unei instanțe a unui obiect, referința sa this indică spre obiectul global. Prin urmare, când am legat bFunc de câmpul b.fun, referința sa nu a fost niciodată actualizată la b însuși. În majoritatea limbajelor de programare care oferă atât stiluri funcționale, cât și orientate-pe-obiect, se face un compromis în modul în care este gestionată auto-referința. JavaScript are abordarea sa, în timp ce Python are o abordare diferită și Scala are încă o abordare diferită.

```
var a = {name: "a", fun: function () { return this; }};
a.fun();
var bFunc = function () { return this }
var b = {name: "b", fun: bFunc};
b.fun();
```

Metaprogramare

O bună definiție a metaprogramării ar fi următoarea: programarea are loc când scrii cod pentru a crea ceva, iar metaprogramarea are loc când scrii cod care schimbă modul în care ceva este interpretat. Cum? De exemplu folosind metoda Function.call pentru a crea noi comportamente.

```
function Point2D(x, y) {
  this._x = x;
  this._y = y;
}
new Point2D(0, 1);
function Point3D(x, y, z) {
  Point2D.call(this, x, y);
  this._z = z;
}
new Point3D(10, -1, 100);
```

Alt exemplu de metaprogramare

Un exemplu destul de comun al metaprogramarii este folosirea proxiurilor pentru jurnalizare sau monitorizare. Aici interceptam prin reflecție apelurile de metode.

```
const target = {
    message1: "hello",
    message2: "everyone"
};
const handler = {
    get: function(target, prop, receiver) {
        console.log(`Accessed ${prop}`);
        return Reflect.get(...arguments);
    }
};
const proxy = new Proxy(target, handler);
console.log(proxy.message1);
console.log(proxy.message2);
```

Map Reduce Filter

```
var nums = [1,2,3,4,5];
function doubleAll(array) {
  return _.map(array, function(n) { return n*2 });
}
doubleAll(nums);
function average(array) {
  var sum = _.reduce(array, function(a, b) { return a+b
      });
  return sum / _.size(array);
}
average(nums);
/* grab only even numbers in nums */
function onlyEven(array) {
  return _.filter(array, function(n) {
    return (n%2) === 0;
  });
}
onlyEven(nums);
```

Exercitii 1

- Se da:

```
var nums = [100,20,25];
```

Creati o functie care sa enumereze toti factorii comuni ai elementelor din sir. Folositi OOP si programarea functionala, comparativ. Testati si timpul de executie.

- Se da:

```
_.find(['a', 'b', 3, 'd'], _.isNumber);
```

Creati exemple care sa verifice egalitatea, daca elementele sunt nule, daca sunt string-uri, etc.

Exercitii 2

- Se da:

```
var albums = [{title: "Sabbath␣Bloody␣Sabbath", genre
    : "Metal"},
          {title: "Scientist", genre: "Dub"},
          {title: "2Pacs", genre: "Hip-Hop"}
          {title: "Undertow", genre: "Metal"}];
```

Creati exemple prin care sa sortati, grupati, numarati pentru diferite genuri. Testati si ideea de intersectie sau reuniune a mai multor albume.