Замикання

- вкладена функція має повний доступу до всіх змінних і функцій, оголошених у зовнішній функції (та інших змінних і функцій, до яких має доступ ця зовнішня функція).
- Однак зовнішня функція не має доступу до змінним і функціям, оголошеним у внутрішній функції.

```
var pet = function (name) {    // Зовнішня функція оголосила змінну "name"
   var getName = function() {
     return name;                          // Вкладена функція має доступ до "name"
зовнішньої функції
   }
   return getName;                         // Повертаємо вкладену функцію, тим самим
зберігаючи доступ
                                // до неї для іншого scope
}
myPet = pet('Vivie');

myPet (); // Повертається "Vivie",
                                           // тому що навіть після виконання зовнішньої
функції
                                           // name зберігся для вкладеної функції
```

https://developer.mozilla.org/ru/docs/Web/JavaScript/Guide/Functions
https://developer.mozilla.org/ru/docs/Web/JavaScript/Closures

Практика

- написати функцію яка буде повертати методи для виведення і задання імені

Модуль

- Ми хочемо винести функцію в окремий файл і підключити до сторінки.
- Оголосили глобальні змінні і наприклад в іншому файлі теж я така змінна
- І в нас виходить конфлікт змінних, щоб цього не було їх потрібно інкапсулювати

```
(function() {

    // код

})();
```

https://learn.javascript.ru/closures-module

this у функціях

- Будь-яка функція може мати в собі this
- this називається контекстом виклику і буде визначено в момент виклику функції

```
function sayHi() {
  alert( this.firstName );
}
```

- Якщо одну і ту ж функцію запускати в контексті різних об'єктів, вона буде отримувати різний this

```
var user = {firstName: "Вася"};
var admin = {firstName: "Адмін"};

function func () {
   alert (this.firstName);
}

user.f = func;
admin.g = func;

// this дорівнює об'єкту перед точкою:
user.f (); // Вася
admin.g (); // Адмін
```

```
var user = {
    firstName: "Вася",
    func: function() {
       alert (this.firstName);
    }
};
var admin = {
    firstName: "Адмін",
    func: function() {
       alert (this.firstName);
    }
};

user.func (); // Вася
admin.func (); // Адмін
```

Практика

- Створіть об'єкт calculator з методами:
- sum () повертає суму цих двох значень
- mul () повертає добуток цих двох значень

```
var calculator = {
          number1: 2,
     number2: 3,
}

alert( calculator.sum() );
alert( calculator.mul() );
```

Об’єкти через new, функції конструктори

- застосовуються коли потрібно створити багато однотипних об’єктів

```
function Animal(name) {
  this.name = name;
  this.canWalk = true;
}

var animal = new Animal("пес");

animal = {
  name: "пес",
  canWalk: true
}
```

https://learn.javascript.ru/constructor-new

Методи в конскукторі

```
function User(name) {
  this.name = name;

  this.sayHi = function() {
    alert( “Hi ” + this.name );
  };
}

var ivan = new User("Іван");

ivan.sayHi(); // Ім’я: Іван
```

Практика

- Переписати об'єкт calculator з методами через конструктор:
- метод sum () повертає суму двох значень
- метод mul () повертає добуток двох значень

```
var calculator = new Calculator();

alert( "Сума=" + calculator.sum() );
alert( "Добуток=" + calculator.mul() );
```

Явне вказування this: "call", "apply"

```
func.call(context, arg1, arg2, ...)
```

Виклик func.call (context, a, b ...) - те саме, що звичайний виклик func (a, b ...), але з явно вказаним this (= context)

```
function showFullName() {
  alert( this.firstName + " " + this.lastName );
}

var user = {
  firstName: "John",
  lastName: "Doe"
};

showFullName.call(user) // "John Doe"
```

https://learn.javascript.ru/call-apply

Із заданими параметрами

```
var user = {
  firstName: "Василь",
  surname: "Петренко",
  patronym: "Іванович"
};

function showFullName(firstPart, lastPart) {
  alert( this[firstPart] + " " + this[lastPart] );
}

// f.call(контекст, аргумент1, аргумент2, ...)
showFullName.call(user, 'firstName', 'surname') // "Василь Петренко"
showFullName.call(user, 'firstName', 'patronym') // "Василь Іванович"
```

func.apply

- Виклик функції за допомогою func.apply працює аналогічно func.call, але приймає масив аргументів замість списку

```
showFullName.apply(user, ['firstName', 'surname']);
```

bind - прив'язка контексту

- Метод bind() створює нову функцію, яка в момент виклику має певне присвоєне значення this , а також задану послідовність аргументів, що передують будь-яким аргументам, переданим під час виклику нової функції

```
var users = {
  data: [
      {name: 'John Smith'},
      {name: 'Ellen Simons'}
  ],

   showFirst: function (event) {
       console.log(this.data[0].name);
   }
}

$('button').click(users.showFirst); // this.data is undefined
```

this в функції clickHandler виступатиме вже не об'єкт users, як ми того хотіли, а об'єкт кнопки, тому і властивість data у нього відсутня

```
$("button").click(users.showFirst.bind(users));
```

- Методи call / apply викликають функцію із заданим контекстом і аргументами.
- А bind не викликає функцію. Він тільки повертає «обгортку», яку ми можемо викликати пізніше, і яка передасть виклик в вихідну функцію, з прив'язаним контекстом

Практика

- Є об'єкт calculator

```
var calculator = {
          number1: 2,
     number2: 3,
}
```

функція sum () повертає суму двох значень
функція mul () повертає добуток двох значень

Явно задати контекст(this) для цих фунцій

Ресурси

https://habr.com/company/ruvds/blog/419371/.com