АРГУМЕНТЫ ФУНКЦИИ

Мы можем передать внутрь функции некоторые данные:

```
1  function greeting() {
2    console.log('Приветствую!');
3  }
4
5  greeting('Иван');
```

Данные мы передаем при вызове функции внутри круглых скобок.

АРГУМЕНТЫ ФУНКЦИИ

Чтобы использовать внешние данные внутри функции нам нужно явно указать, что функция может принимать ***аргументы***:

```
1 function greeting(name) {
2   console.log(`Приветствую, ${name}!`);
3 }
4
5 greeting('Иван');
```

В круглых скобках при объявлении функции пишем имя переменной, которую сможем использовать внутри функции.

ЗНАЧЕНИЕ АРГУМЕНТОВ ПО УМОЛЧАНИЮ

Если мы не передадим требуемый аргумент при вызове функции:

```
1 function greeting(name) {
2   console.log(`Приветствую, ${name}!`);
3 }
4
5 greeting();
```

То в консоли при выполнении файла увидим:

```
Приветствую, undefined!
```

ЗНАЧЕНИЕ АРГУМЕНТОВ ПО УМОЛЧАНИЮ

В **ES2015** появилась возможность задавать значение по умолчанию при определение функции:

```
1  function greeting(name = 'Незнакомец') {
2    console.log(`Приветствую, ${name}!`);
3  }
4
5  greeting(); // Приветствую, Незнакомец!
6  greeting(undefined); // Приветствую, Незнакомец!
7  greeting(null); // Приветствую, null!
8  greeting(false); // Приветствую, false!
9  greeting(0); // Приветствую, 0!
```

ВОЗВРАТ РЕЗУЛЬТАТА

Для проброса результата за пределы функции нужно использовать ключевое слово return внутри функции когда мы хотим вернуть данные.

```
1  function summ(a, b) {
2     return a + b;
3  }
4
5  var result = summ(5, 3);
```

При этом вычисляется выражение справа и его значение возвращается как результат вызова функции.

ВОЗВРАТ РЕЗУЛЬТАТА

После вызова return последующие команды внутри функции не выполняются:

```
function summ(a, b) {
  return a + b;
  console.log('Не будет выведен');
}

var result = summ(5, 3);
```

ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ

Переменные, указанные как аргументы функции, доступны только внутри функции и не видны за её пределами:

```
1  function log(message) {
2console.log(message);
3  }
4
5  console.log(message); // undefined
6  log('Сообщение'); // Сообщение
7  console.log(message); // undefined
```

ВНЕШНИЕ ПЕРЕМЕННЫЕ

Но если внутри функции объявлена переменная с тем же именем, то интерпретатор не будет искать ее за пределами функции и, как следствие, доступа к внешней переменной не будет.

```
1  var name = 'Иван';
2  function greeting() {
3    var name = 'Олег';
4    console.log(`Приветствую, ${name}!`);
5  }
6
7  greeting(); // Приветствую, Олег!
```

ВНЕШНИЕ ПЕРЕМЕННЫЕ

И аналогично в **ES5**:

```
1  var name = 'Иван';
2  function greeting(name) {
3    console.log(`Приветствую, ${name}!`);
4  }
5
6  greeting(); // Приветствую, undefined!
```

Не смотря на одинаковые имена, для интерпретатора это разные переменные, созданные в разных областях памяти.

ФУНКЦИОНАЛЬНОЕ ВЫРАЖЕНИЕ

Между переменными и функциями есть некоторое сходство в поведении. Что будет, если использовать функцию как переменную? Выведем функцию а консоль:

```
1  function greeting(name) {
2    console.log(`Приветствую, ${name}!`);
3  }
4
5  console.log(greeting);
```

ФУНКЦИОНАЛЬНОЕ ВЫРАЖЕНИЕ

Или присвоим функцию как значение переменной:

```
1  function greeting(name) {
2    console.log(`Приветствую, ${name}!`);
3  }
4
5  var hello = greeting;
6  hello('Иван'); // Приветствую, Иван!
```

```
var greeting = function (name) {
  console.log(`Приветствую, ${name}!`);
};

greeting('Иван'); // Приветствую, Иван!
```

ЛИТЕРАЛ ФУНКЦИОНАЛЬНОГО ВЫРАЖЕНИЯ

Можно не сохранять функцию в переменную, а сразу вызвать (выполнить) ее:

```
1 (function (name) {
2   console.log(`Приветствую, ${name}!`);
3 })('Иван'); // Приветствую, Иван!
```

Результатом такого кода будет одноразовая функция. После выполнения она станет недоступна. В JS одноразовые функции – частая практика.

ИМЕНОВАННОЕ ФУНКЦИОНАЛЬНОЕ ВЫРАЖЕНИЕ

Функциональному выражению можно присвоить имя:

```
var greeting = function hello(name) {
  console.log(`Приветствую, ${name}!`); 3
};

greeting('Иван'); // Приветствую, Иван!
hello('Иван'); // Ошибка
```

Если у функционального выражения нет имени, то мы не можем вызвать его внутри самой функции. Рекурсия исключена.

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

Ранее мы узнали, что функцию можно присвоить в переменную. Так же функцию можно поместить в элемент массива:

```
1 var operations = [];
2 operations[0] = function (a, b) {
3   return a + b;
4 }
```

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

Аналогично можно сделать при помощи метода push :

```
1  var operations = [];
2  operations.push(function (a, b) {
3    return a + b; 4
});
5 operations.push(function (a, b) {
6    return a / b; 7
});
```

Или при создании массива:

```
1  var operations = [function (a, b) {
2    return a + b;
3  }, function (a, b) {
return a / b;
```

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

Можем передавать функцию в другую функцию:

```
1  function calc(a, b, op) {
2return op(a, b);
3  }
4
5  function summ(a, b) {
6    return a + b;
7  }
8
9  var result = calc(11, 31, summ);
10 console.log(result); // 42
```

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

А что если функция, созданная внутри функции, потребуется нам за ее пределами? Можем вернуть ее в качестве результата.

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

  return summ;
}

var operation = createOperation();
var result = operation(9, 33);
console.log(result); // 42
```