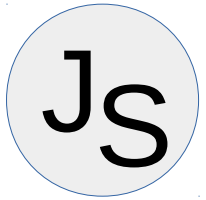
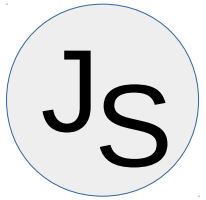
# Переменные в JavaScript




- Типизация и декларация
- Область видимости
- Блочная область видимости
- Примитивные и ссылочные типы
- Сборщик мусора
- Переменные и свойства объекта
- Цепочка видимости (scope chain)

# Типизация

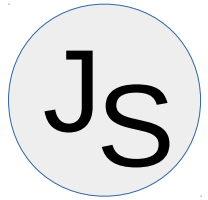



JavaScript – нетипизированный язык

```
i = 10;
i = "десять";
```

\+ простота
\- нет строгости, ненадёжность

# Объявление переменной




- Явно

```
var i=10;
var n, sum;
var message='Привет';
var i=0, j=0, k;          // undefined
```

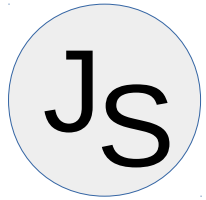
- Неявно (глобальные)

```
j = "десять";
```

Допускаются повторные объявления

```
var j=0;
var j=1;
```

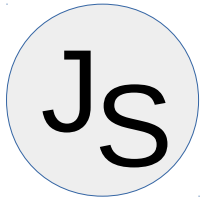
# Область видимости



```
var scope = "global";           // Глобальная переменная
function checkscope( ) {
    var scope = "local";        // Локальная переменная
                                // с тем же именем
    document.write(scope);      // Используется локальная
}
checkscope( );                  // Выводит "local"
```

```
scope = "global";                 // Глобальная переменная
function checkscope( ) {
    scope = "local";              // Изменили глобальную!
    console.log(scope);           // Глобальная
    myscope = "local";            // Неявное объявление
                                  // глобальной переменной
    console.log(myscope);         // Глобальная
}
checkscope( );                    // Выводит "locallocal"
console.log(scope);               // Выводит "local"
console.log(myscope);             // Выводит "local"
```

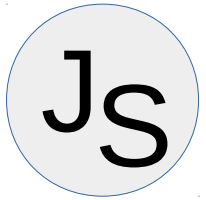
# Область видимости




- Объявленная с помощью var переменная действует во всей функции

```
function test(o) {
    var i = 0;              // i действует во всей функции
    if (typeof o == "object") {
        var j = 0;          // j действует не только в блоке
        for(var k = 0; k < 10; k++) {
            console.log(k);
        }
        console.log(k);     // k определена, выводит 10
    }
    console.log(j);         // j определена, возможно undefined
}
```

# Блочная видимость (ES 2015)



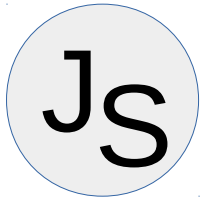

- var - действует во всей функции

```
var apples = 5;

if (true) {
  var apples = 10; // повторное объявление
  alert(apples);   // 10 (внутри блока)
}

alert(apples);     // 10 (снаружи блока то же самое)
```

- let - действует в пределах блока

```
let apples = 5;

if (true) {
  let apples = 10;// новая переменная
  alert(apples);  // 10 (внутри блока)
}

alert(apples);    // 5 (снаружи блока значение не менялось)
```

# Блочная видимость (ES 2015)




- var – видна во всей функции

```
alert(a); // undefined

var a = 5;
```

- let - видна только после объявления

```
alert(a); // ошибка, нет такой переменной

let a = 5;
```

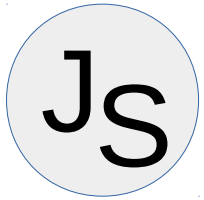
- Повторное объявление не разрешается

```
let x;
let x; // ошибка: переменная x уже объявлена
```

# Задача "Армия функций"

```
function makeArmy() {

  let shooters = [];

  for (let i = 0; i < 10; i++) {
    shooters.push(function() {
      alert( i ); // выводит свой номер
    });
  }

  return shooters;
}

var army = makeArmy();

army[0](); // 0
army[5](); // 5
```

# Константы (ES 2015)

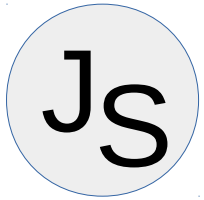



Аналогично let, но нельзя менять значение

```
const apple = 5;
apple = 10; // ошибка
```

```
const user = {
  name: "Вася"
};

user.name = "Петя"; // допустимо
user = 5; // нельзя, будет ошибка
```

# Примитивные и ссылочные типы




## Примитивные

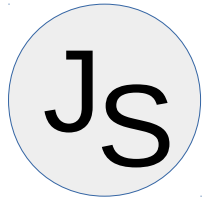
- Число
- Логическое значение
- null
- undefined

## Ссылочные

- Объект
- Массив
- Функция

## Символьная строка

# Примитивные и ссылочные типы



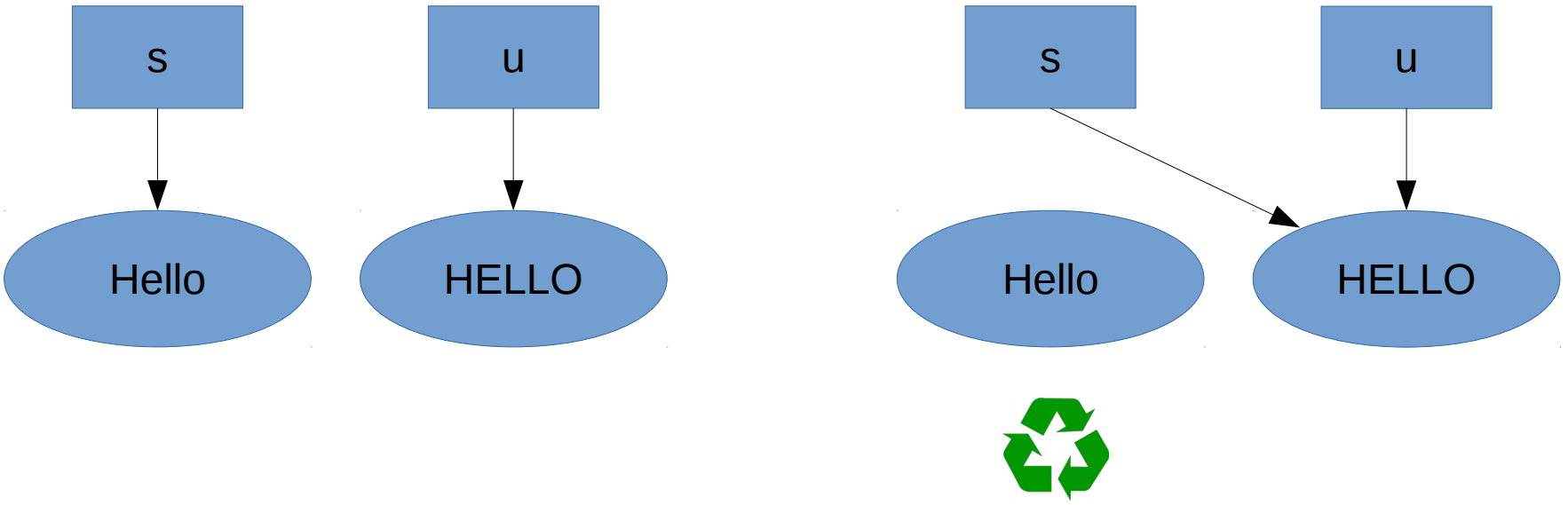

```
var a = 3.14;
var b = a;
a = 4;
alert(b);          // 3.14
```

```
var a = [1,2,3];
var b = a;
a[0]  = [99];
alert(b);          // [99,2,3]
```

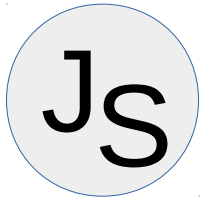
# Сборка мусора

Garbage collection

```
var s = "hello";
var u = s.toUpperCase( );
s = u;
```



# Переменные и свойства




Принципиальной разницы нет

Глобальный объект

- this.parseInt()
- this.Infinity
- window.Math
- window.location

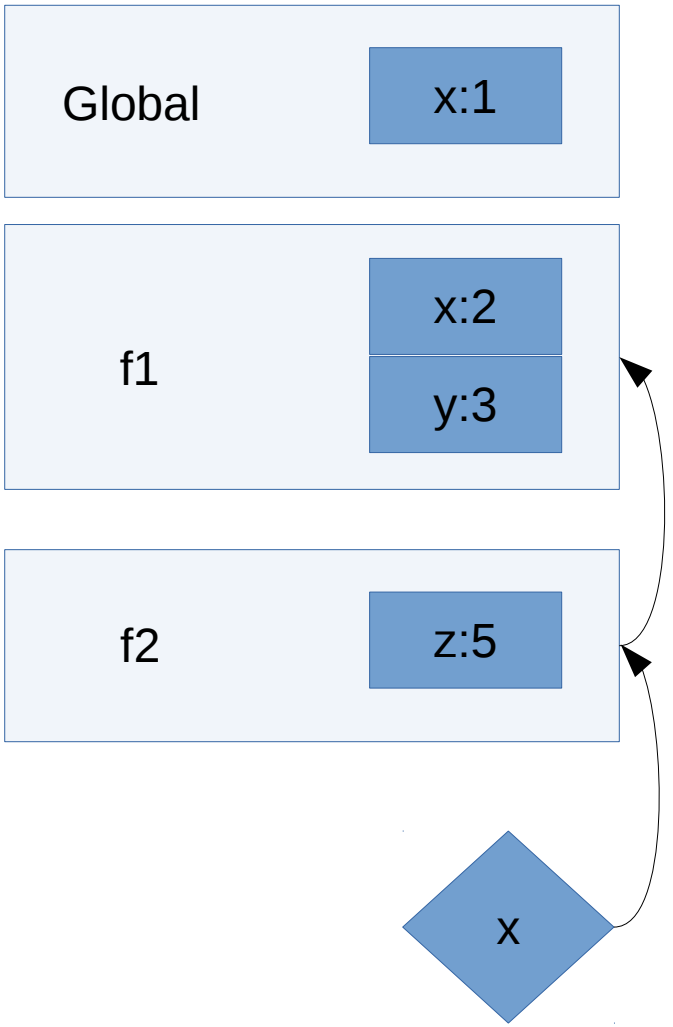
Объект вызова (call object)

- Локальные переменные и параметры функции

Контекст выполнения (функция, глобальный)

# Цепочка видимости (scope chain)

```
var x=1;

function f1(){

   var x=2,y=3;

   function f2(){
      var z=5;
      alert(x);
   }

}
```



# Вопрос

- Что будет выведено?

```
var scope = "global";
function f( ) {
    alert(scope);
    var scope = "local";
    alert(scope);
}
f( );
```

- Что будет, если var заменить на let?
- Что будет, если var заменить на const?

# Переменные в JavaScript

- Типизация и декларация
- Область видимости
- Блочная область видимости
- Примитивные и ссылочные типы
- Сборщик мусора
- Переменные и свойства объекта
- Цепочка видимости (scope chain)

# Типизация

JavaScript – нетипизированный язык

```
i = 10;
i = "десять";
```

+ простота
- нет строгости, ненадёжность

# Объявление переменной

- Явно

```
var i=10;
var n, sum;
var message='Привет';
var i=0, j=0, k;        // undefined
```

- Неявно (глобальные)

```
j = "десять";
```

Допускаются повторные объявления

```
var j=0;
var j=1;
```

# Область видимости

```
var scope = "global";          // Глобальная переменная
function checkscope( ) {
    var scope = "local";       // Локальная переменная
                               // с тем же именем
    document.write(scope);     // Используется локальная
}
checkscope( );                 // Выводит "local"
```

```
scope = "global";                // Глобальная переменная
function checkscope( ) {
    scope = "local";             // Изменили глобальную!
    console.log(scope);          // Глобальная
    myscope = "local";           // Неявное объявление
                                 // глобальной переменной
    console.log(myscope);        // Глобальная
}
checkscope( );                   // Выводит "locallocal"
console.log(scope);              // Выводит "local"
console.log(myscope);            // Выводит "local"
```

# Область видимости

- Объявленная с помощью var переменная действует во всей функции

```
function test(o) {
    var i = 0;            // i действует во всей функции
    if (typeof o == "object") {
        var j = 0;        // j действует не только в блоке
        for(var k = 0; k < 10; k++) {
            console.log(k);
        }
        console.log(k);   // k определена, выводит 10
    }
    console.log(j);       // j определена, возможно undefined
}
```

# Блочная видимость (ES 2015)

- var - действует во всей функции

```
var apples = 5;

if (true) {
  var apples = 10; // повторное объявление
  alert(apples);   // 10 (внутри блока)
}

alert(apples);     // 10 (снаружи блока то же самое)
```

- let - действует в пределах блока

```
let apples = 5;

if (true) {
  let apples = 10;// новая переменная
  alert(apples);  // 10 (внутри блока)
}

alert(apples);    // 5 (снаружи блока значение не менялось)
```

# Блочная видимость (ES 2015)

- var – видна во всей функции

```
alert(a); // undefined

var a = 5;
```

- let - видна только после объявления

```
alert(a); // ошибка, нет такой переменной

let a = 5;
```

- Повторное объявление не разрешается

```
let x;
let x; // ошибка: переменная x уже объявлена
```

# Задача "Армия функций"

```
function makeArmy() {

  let shooters = [];

  for (let i = 0; i < 10; i++) {
    shooters.push(function() {
      alert( i ); // выводит свой номер
    });
  }

  return shooters;
}

var army = makeArmy();

army[0](); // 0
army[5](); // 5
```

# Константы (ES 2015)

Аналогично let, но нельзя менять значение

```
const apple = 5;
apple = 10; // ошибка
```

```
const user = {
  name: "Вася"
};

user.name = "Петя"; // допустимо
user = 5; // нельзя, будет ошибка
```

## Примитивные и ссылочные типы

### Примитивные

- Число
- Логическое значение
- null
- undefined

### Ссылочные

- Объект
- Массив
- Функция

### Символьная строка

A primitive type has a fixed size in memory. For example, a number occupies eight bytes of memory, and a boolean value can be represented with only one bit.

Reference types are another matter, however. Objects, for example, can be of any length -- they do not have a fixed size. The same is true of arrays: an array can have any number of elements. Similarly, a function can contain any amount of JavaScript code. Since these types do not have a fixed size, their values cannot be stored directly in the eight bytes of memory associated with each variable. Instead, the variable stores a reference to the value. Typically, this reference is some form of pointer or memory address.

For efficiency, we would expect JavaScript to copy references to strings, not the actual contents of strings. On the other hand, strings behave like a primitive type in many ways.

## Примитивные и ссылочные типы

```
var a = 3.14;
var b = a;
a = 4;
alert(b);        // 3.14
```

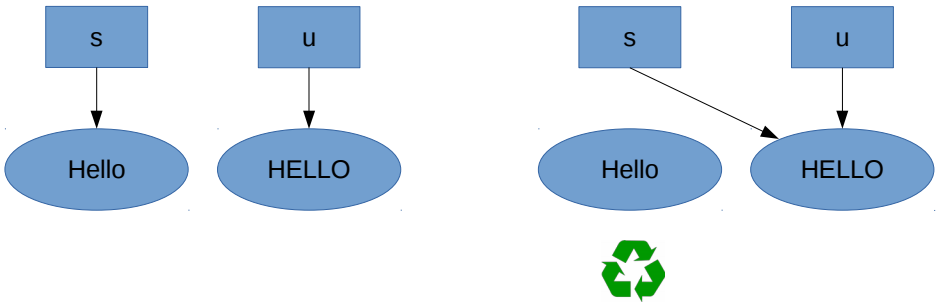
```
var a = [1,2,3];
var b = a;
a[0]  = [99];
alert(b);        // [99,2,3]
```

The question of whether strings are a primitive or reference type is actually moot, because strings are immutable: there is no way to change the contents of a string value.

# Сборка мусора

Garbage collection

```
var s = "hello";
var u = s.toUpperCase( );
s = u;
```



# Переменные и свойства

Принципиальной разницы нет

Глобальный объект

- this.parseInt()
- this.Infinity
- window.Math
- window.location

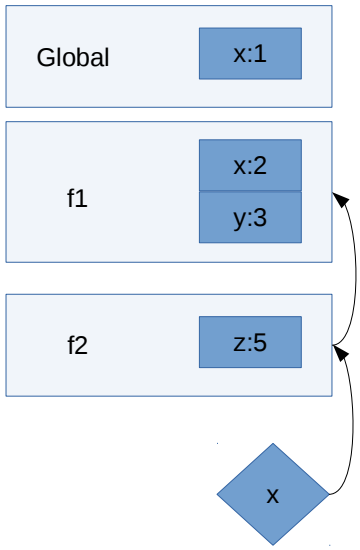
Объект вызова (call object)

- Локальные переменные и параметры функции

Контекст выполнения (функция, глобальный)

# Цепочка видимости (scope chain)

```
var x=1;

function f1(){

   var x=2,y=3;

   function f2(){
      var z=5;
      alert(x);
   }

}
```



# Вопрос

- Что будет выведено?

```
var scope = "global";
function f( ) {
    alert(scope);
    var scope = "local";
    alert(scope);
}
f( );
```

- Что будет, если var заменить на let?
- Что будет, если var заменить на const?