

JS: Functions types, hoisting

НАШИ ПРАВИЛА



Включенная камера



Вопросы по поднятой руке



Не перебиваем друг друга



Все вопросы, не связанные с тематикой курса (орг-вопросы и т. д.), должны быть направлены куратору



Подготовьте свое рабочее окружение для возможной демонстрации экрана (закройте лишние соцсети и прочие приложения)

Повторим;)

■ Как вызвать функцию?

■ Что такое параметры функции? Для чего они нужны?

■ Каким образом мы можем вернуть значение из функции?

ЦЕЛЬ

Изучить различные типы функций и поднятие переменных и функций

ПЛАН ЗАНЯТИЯ

- Function types
- Область видимости переменных
- Hoisting

Function types

() => {}

Существует ещё один синтаксис создания функций, который называется **Function Expression** (Функциональное Выражение).

Данный синтаксис позволяет нам создавать новую функцию в середине любого выражения.

```
1 let sayHi = function() {  
2   alert( "Привет" );  
3 };
```



Function Expression

```
1 function sayHi() {  
2   alert( "Привет" );  
3 }
```



Function Declaration

Разница Function Expression и Function Declaration

1. **Function Declaration** может быть вызвана раньше, чем она объявлена.

```
1 sayHi("Вася"); // Привет, Вася
2
3 function sayHi(name) {
4     alert( `Привет, ${name}` );
5 }
```

1. **Function Expression** создаётся, когда выполнение доходит до него, и затем уже может использоваться.

```
1 sayHi("Вася"); // ошибка!
2
3 let sayHi = function(name) { // (*) магии больше нет
4     alert( `Привет, ${name}` );
5 };
```


Стрелочные функции

Существует ещё один очень простой и лаконичный синтаксис для создания функций. Он называется «функции-стрелки» или «стрелочные функции» (arrow functions), т.к. выглядит следующим образом:

```
1 let func = (arg1, arg2, ...argN) => expression;
```

Это создаёт функцию func, которая принимает аргументы arg1..argN, затем вычисляет expression в правой части с их использованием и возвращает результат.

Обычная функция

```
1 let func = function(arg1, arg2, ...argN) {  
2   return expression;  
3 };
```

Стрелочные функции

Если у нас только один аргумент, то круглые скобки вокруг параметров можно опустить, сделав запись ещё короче:

```
1 let double = n => n * 2;  
2 // примерно тоже что и: let double = function(n) { return n * 2 }  
3  
4 alert( double(3) ); // 6
```

Многострочные стрелочные функции

Иногда нам нужна более сложная функция, с несколькими выражениями и инструкциями. Это также возможно, нужно лишь заключить их в фигурные скобки. При этом важное отличие – в том, что в таких скобках для возврата значения нужно использовать `return` (как в обычных функциях).

```
let sum = (a, b) => {
```

```
  let result = a + b;
```

```
  // если мы используем фигурные скобки, то нам нужно явно указать  
  "return"
```

```
    return result;
```

```
};
```

```
alert( sum(1, 2) );
```

Функция в качестве параметра

Функция может передаваться в качестве аргумента при вызове другой функции. Например, функция, которая может выполнить произвольную операцию между двумя числами.

```
function operateOnNumbers(a, b, operation) {  
  return operation(a, b);  
}
```

```
// Функция сложения  
function add(x, y) {  
  return x + y;  
}
```

```
// Функция вычитания  
function subtract(x, y) {  
  return x - y;  
}
```

```
const sumResult = operateOnNumbers(5, 3, add);  
console.log(sumResult);
```

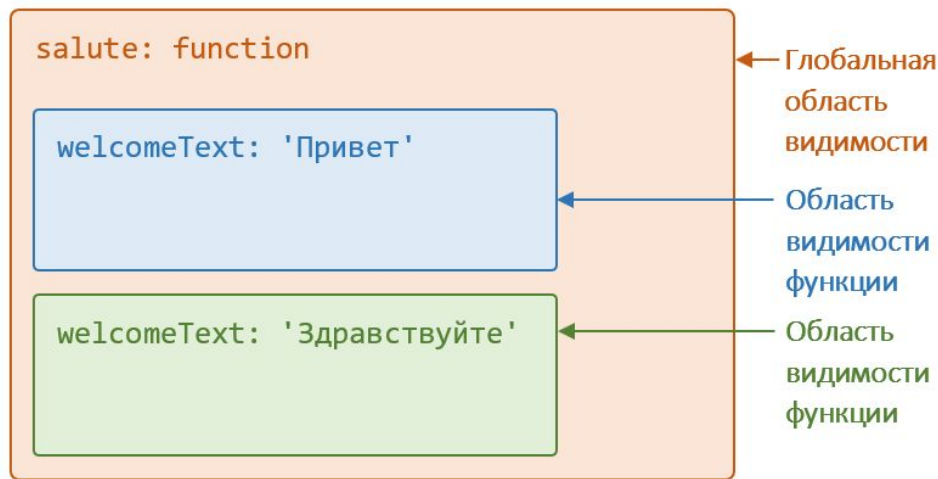
```
const differenceResult = operateOnNumbers(8, 3, subtract);  
console.log(differenceResult);
```

Области видимости переменных



Область видимости определяет, где в коде программы будут доступны переменные и функции. В JavaScript есть два типа области видимости — **глобальная** и **локальная**

Переменные `let` и `const` ведут себя аналогично, в примерах будем употреблять `let`



Блоки кода

Если переменная объявлена внутри блока кода {...}, то она видна только внутри этого блока.

```
1 {  
2   // выполняем некоторые действия с локальной переменной, которые не должны  
3  
4   let message = "Hello"; // переменная видна только в этом блоке  
5  
6   alert(message); // Hello  
7 }  
8  
9 alert(message); // ReferenceError: message is not defined
```

Для **if**, **for**, **while** и т.д. переменные, объявленные в блоке кода {...}, также видны только внутри:

```
1  for (let i = 0; i < 3; i++) {  
2    // переменная i видна только внутри for  
3    alert(i); // 0, потом 1, потом 2  
4  }  
5  
6  alert(i); // Ошибка, нет такой переменной!
```


Устаревшее ключевое слово `var`



Устаревшее ключевое слово "var".

Обычно var не используется в современных скриптах, но всё ещё может скрываться в старых.

На первый взгляд, поведение var похоже на let. Например, объявление переменной:

```
1 function sayHi() {  
2   var phrase = "Привет"; // локальная переменная, "var" вместо "let"  
3  
4   alert(phrase); // Привет  
5 }  
6  
7 sayHi();  
8  
9 alert(phrase); // Ошибка: phrase не определена
```

Для «**var**» не существует блочной области видимости

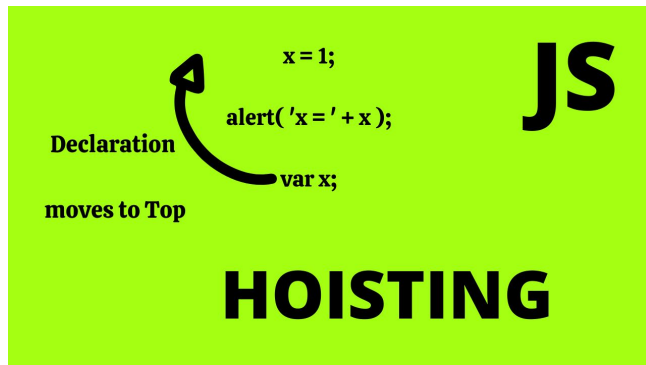
Область видимости переменных **var** ограничивается либо функцией, либо, если переменная глобальная, то скриптом. Такие переменные доступны за пределами блока (if, for, while и т.д).


```
1  if (true) {  
2    var test = true; // используем var вместо let  
3  }  
4  
5  alert(test); // true, переменная существует вне блока if
```

Так как var игнорирует блоки, мы получили глобальную переменную test.

Поднятие или **hoisting** — это механизм в JavaScript, в котором переменные и объявления функций, передвигаются вверх своей области видимости перед тем, как код будет выполнен.

Это позволяет вам использовать переменные и функции до их явного объявления в коде.





Ваша новая IT-профессия – Ваш новый уровень жизни

Программирование с нуля в
немецкой школе AIT TR GmbH