



Лекция 3

Функции

Цели на сегодня

- Начать использовать строгий режим;
- Разобраться с синтаксисом функций;
- Стрелочные функции;
- Функциональные выражения;
- Область видимости;
- IIFE.

Типы данных (Data Types)

1. Прimitives

- a. Числа (Numbers);
- b. Строки (Strings);
- c. Логические (Boolean)
- d. Символ (Symbol);
- e. null;
- f. undefined.

2. Сложные

- a. Объекты (Objects);
- b. Массивы (Arrays);
- c. Функции (Functions).

Функции (Functions) –

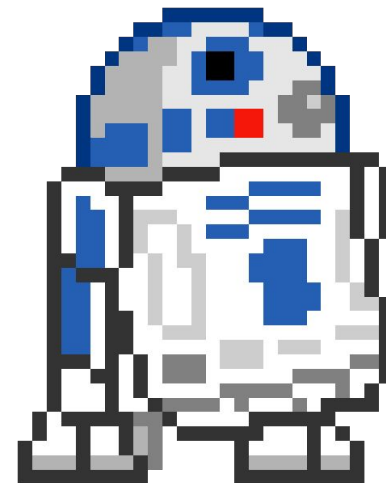
это лучший способ повторять один и тот же код в разных частях программы.

Мы имеем какие-то данные и хотим произвести над ними действия.
Мы знаем, что эти действия придется выполнять часто, поэтому
создаем для этого помощника.

Для начала мы научим его выполнять простые
расчеты, постепенно усложняя его ПО.

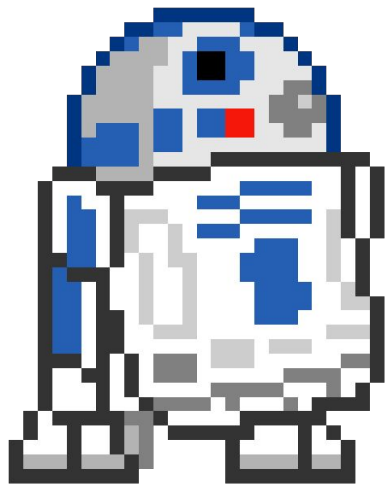
Пусть сегодня он просто будет рассчитывать
площадь поверхности сферы по формуле **$S=4\pi R^2$** .

В домашнем задании к четвертому занятию вы
научите его большому количеству полезностей.



R2D2()

Указываем
радиус



Получаем
результат

Синтаксис функций



// Функцию можно объявить вот так:

```
function имя_функции(параметры) {  
    тело_функции  
}
```

/*

При этом, после объявления функция не выполнится,
а просто запишется в память.

Вызывается функция таким образом:

*/

```
имя_функции(параметры); // Какой-то результат  
имя_функции(другие_параметры); // Другой результат
```




```
/*
```

Функции – конструкции, которые позволяют нам переиспользовать код, повторять одно и то же действие в разных частях программы.

```
*/
```

```
function addThree(number) {  
    return number + 3;  
}
```

// Вызов функции происходит так:

```
addThree(10); // 13  
addThree(5); // 8  
addThree(100); // 103
```

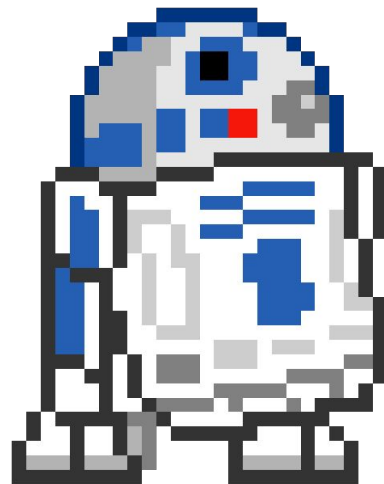
Параметры (аргументы)



```
/*  
    При вызове функции ей можно передать данные, которые та использует  
    по своему усмотрению. Параметры записываются во внутренние переменные  
    функции.  
*/  
  
function greeting(name, text) {  
    const greetText = `Hello, ${name}, there is a message for you: ${text}.`;   
    console.log(greetText);  
}  
  
greeting('Masha', 'You are so cute :3');
```

Самостоятельная работа

1. Напишите функцию `square(x)`, которая считает квадрат переданного из prompt числа, и возвращает в консоль значение "Квадрат числа **x** равен **y**".
2. Если ничего не передано, то возвращает в консоль значение "Число не передано".
3. Если NaN, то возвращает "Значение **x** не является числом".



R2D2()



```
/*
```

```
    Параметров можно передавать больше или меньше, чем необходимо.  
    При этом никакой ошибки не будет.
```

```
*/
```

```
function greeting(name, text, login) {  
    const greetText = `Hello, ${name}, there is a message for you: ${text}.  
                        Your login is ${login}.`;   
    console.log(greetText);  
}
```

```
greeting('Masha', 'You are so cute :3');  
// ...Your login is undefined.
```

```
/*  
    Если нам не нравится видеть в сообщении 'undefined', то мы можем  
    задать значение по умолчанию.  
*/  
  
function greeting(name, text, login = 'user') {  
    const greetText = `  
        Hello, ${name}, there is a message for you: ${text}.  
        Your login is ${login}.`;  
  
    console.log(greetText);  
}  
  
greeting('Masha', 'You are so cute :3'); // ...Your login is user.
```

ES6


```
/*  
    Параметры функции можно получить при помощи оператора ...someName (rest)  
    Через несколько занятий вы научитесь удобно с ними работать  
*/  
  
const makeSomeNoize = (...args) => {  
    alert(args);  
    return args;  
}  
  
makeSomeNoize('Test', 'Some sound', 23, true); // ["Test", "Some sound", 23, true]
```

Возврат значения (return)

return –

используется для того, чтобы функция возвращала значение.

Функции могут иметь директиву `return`, а могут и не иметь. Если `return` не указан или указан без значения, то возвращается `undefined`. При этом никакой ошибки не возникает.



```
function square(num) {  
    num * num; // (0)  
}  
square(2); // Вернет undefined
```

```
function square(num) {  
    return num * num; // (1)  
}  
square(2); // Вернет 4
```

```
function square(num) {  
    console.log(num * num); // (2)  
}  
square(3); // Вернет undefined, но console.log вернет '9'
```



```
/*  
    return завершает выполнение функции. Никакой код после этого не выполнится.  
*/  
  
function add(a, b) {  
  
    return +a + +b;  
  
    // Код ниже никогда не выполнится  
  
    console.log('Test');  
    console.log('Why it is not working? :C');  
}  
  
add(2,10); // unreachable code after return statement
```



// Если функция возвращает значение, его можно записать в переменную

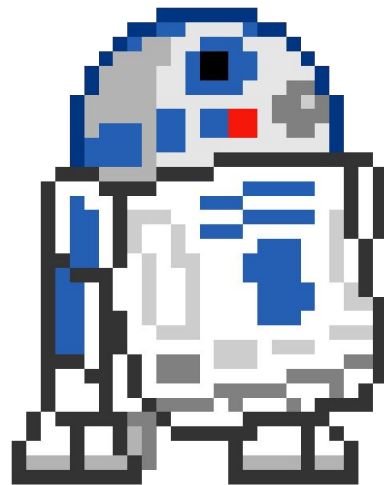
```
let result; // undefined
```

```
function discriminant(a, b, c) {  
    return b * b - 4 * a * c;  
}
```

```
result = discriminant(2, 10, 10);  
console.log(result); // 20
```

Самостоятельная работа

1. Напишите функцию `R2D2()`, которая принимает в качестве параметра радиус сферы и возвращает значение в консоль. Формула:
 $S=4\pi R^2$.
2. Напишите функцию `greetings()`, которая в зависимости от переданного значения (`user`, `admin` или ничего не передано) будет по-разному приветствовать пользователя.



`R2D2()`

Стрелочные функции (Arrow functions)

ES6

/*

Намного чаще сейчас используется синтаксис стрелочных функций.
У стрелочных функций убирается слово `function` и добавляется
стрелка (`=>`).

*/

// Раньше

```
const addTwo = function(num) {  
  return +num + 2;  
}
```

// Сейчас

```
const addTwo = (num) => {  
  return +num + 2;  
}
```

// Или еще компактнее

```
const addTwo = (num) => +num + 2;
```

Область видимости



// Тут все переменные глобальные. Тут не видно ничего из внутренних областей

```
const number = 25;
```

```
function test() {
```

```
    // Тут все переменные локальные
```

```
    const another = 10;
```

```
    console.log(number);
```

```
}
```

```
test(); // 25
```

```
console.log(another); // another is not defined
```

Немедленно вызываемые функции (IIFE)

Немедленно вызываемая функция (IIFE) –

это синтаксическая конструкция, позволяющая вызвать функцию сразу же в месте ее определения.



// Немедленно вызываемая функция выглядит так:

```
(function () {  
    // Тело функции  
})();
```

// Или так:

```
(function () {  
    // Тело функции  
})();
```

Функциональные выражения



// Объявление Function Declaration

```
function addThree(num) {  
    return num + 3;  
}
```

// Объявление Function Expression

```
const addThree = function(num) {  
    return num + 3;  
}
```



```
/*
```

Функции, созданные с помощью FD создаются сразу,
поэтому их можно вызывать раньше объявления

```
*/
```

```
addThree(2); // 5
```

```
function addThree(num) {  
    console.log(num + three);  
}
```

Такое поведение называют
термином “поднятие” (hoisting)



// Функции, созданные с помощью FE нельзя вызвать до оглашения

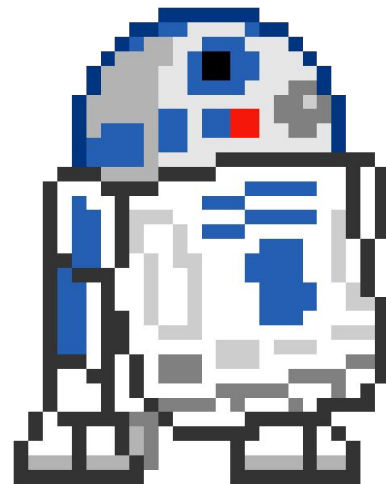
`addThree(2);` // ReferenceError

```
const addThree = function(num) {  
    console.log(num + three);  
}
```

`addThree(2);` // 5

Самостоятельная работа

1. Напишите стрелочную самовызывающуюся функцию, которая принимает два числа и возвращает меньшее из них.

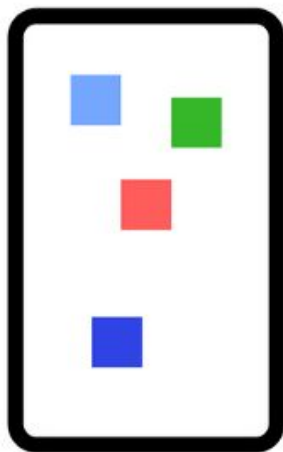


R2D2()

Немного о памяти

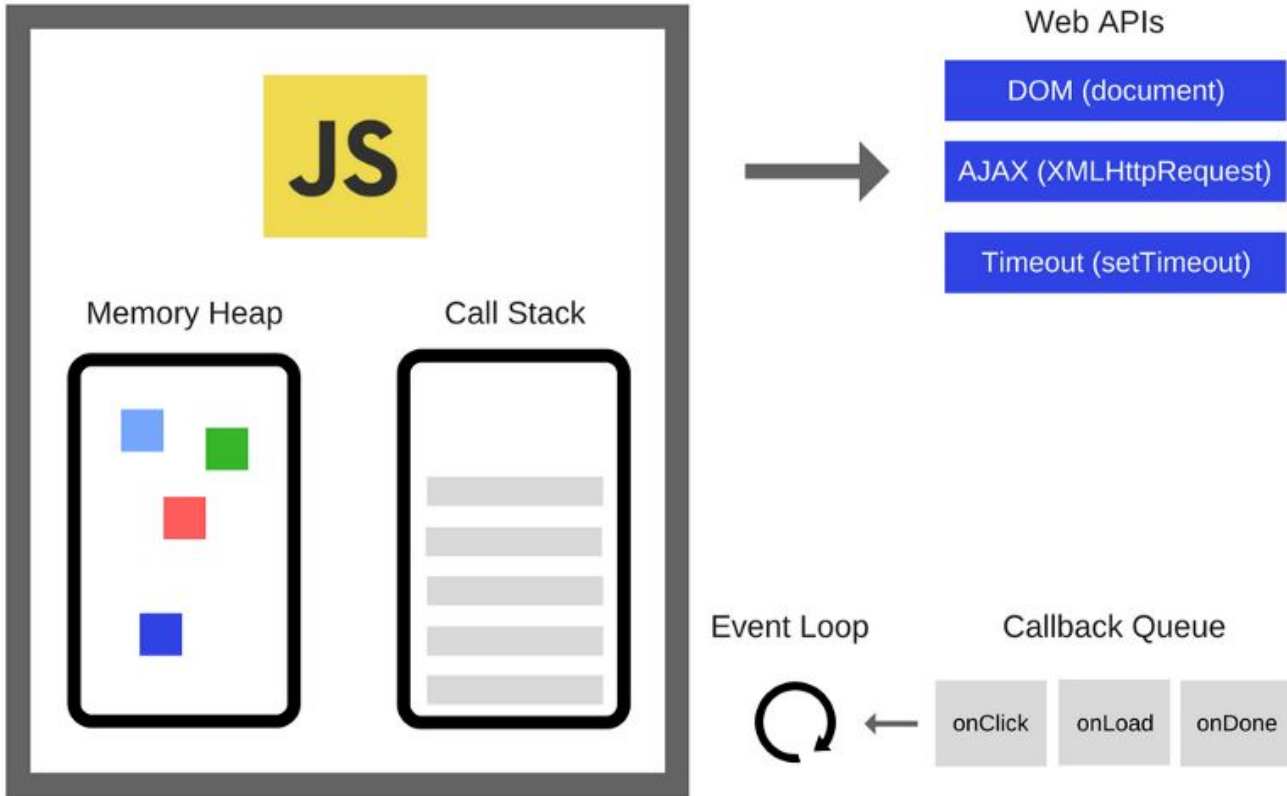



Memory Heap



Call Stack





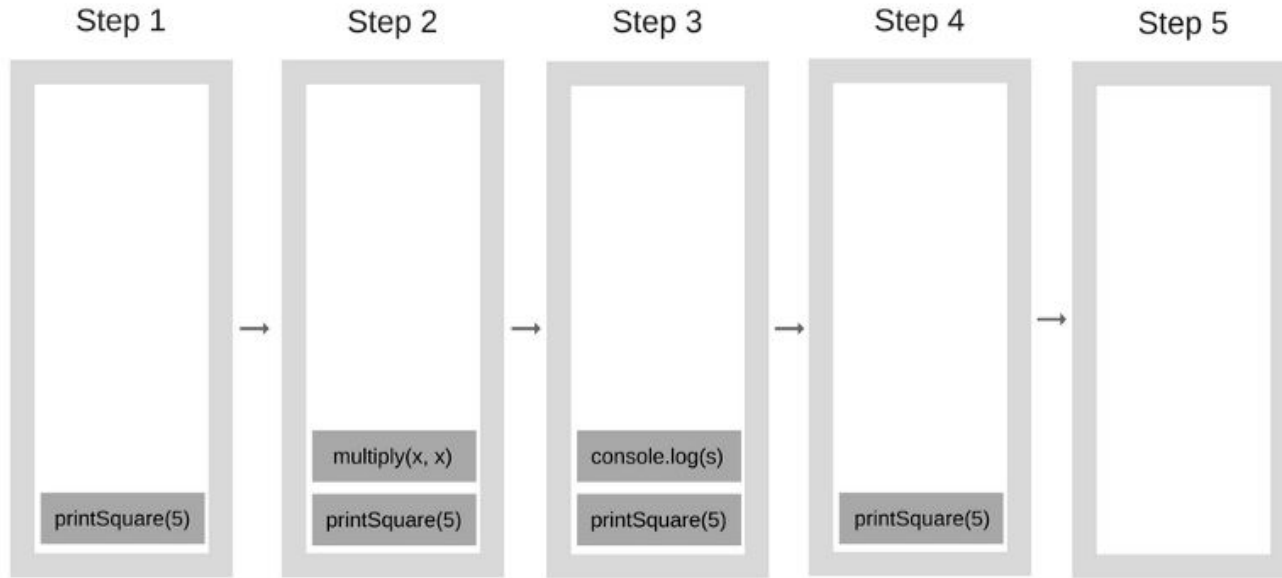


```
const multiply = (x, y) => {  
  return x * y;  
}
```

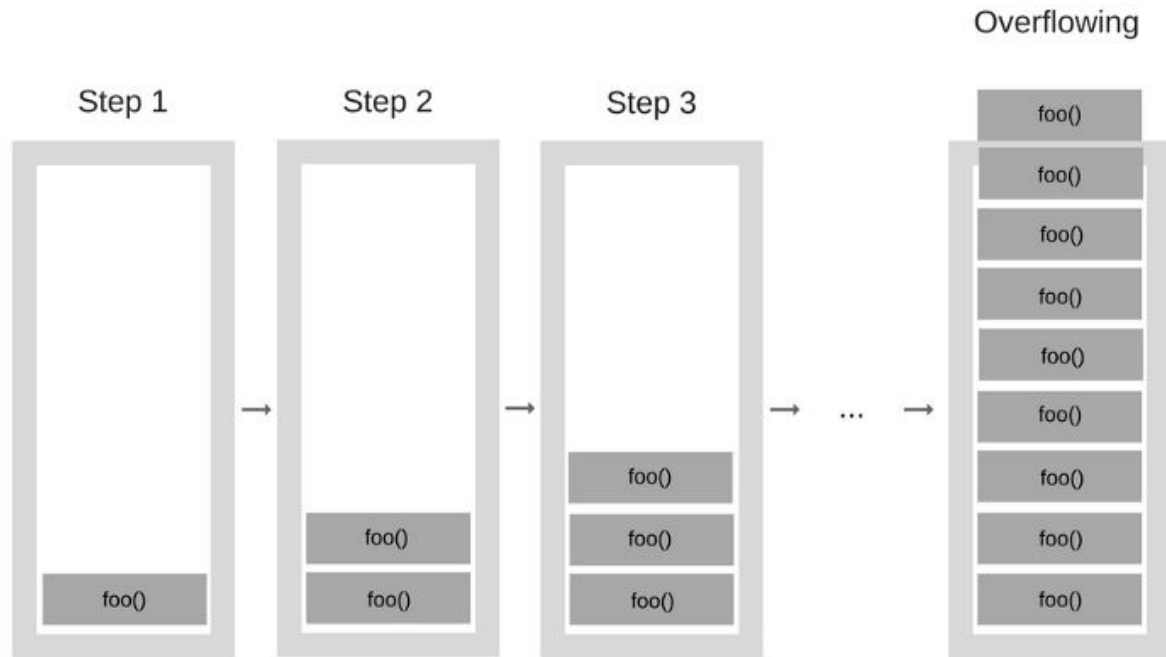
```
const printSquare = (x) => {  
  const s = multiply(x, x);  
  console.log(s);  
}
```

```
printSquare(5);
```

Call Stack



Call Stack



Рекурсия

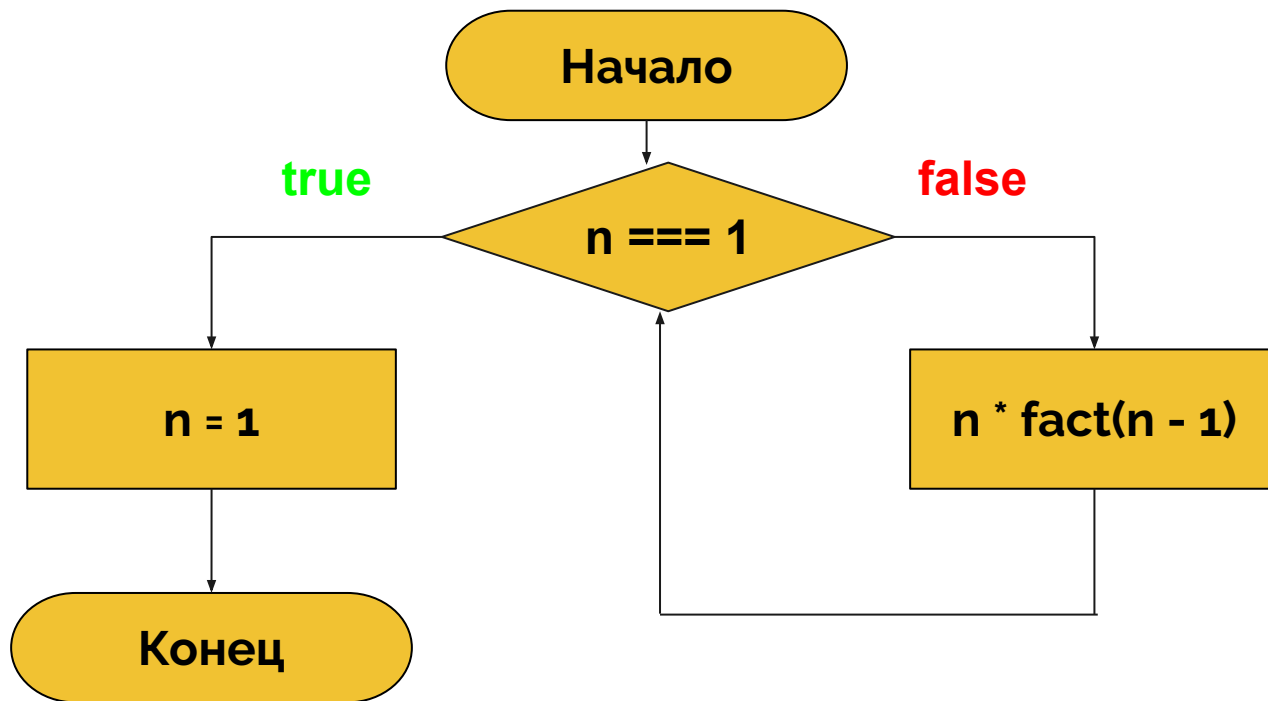
В теле функции могут быть вызваны другие функции для выполнения подзадач. Частный случай такого вызова – когда функция вызывает сама себя. Это называется рекурсией.

Базовый и рекурсивный случаи

Так как рекурсивная функция вызывает сама себя, программисту легко ошибиться и написать функцию так, что возникнет бесконечный цикл.

Когда вы пишете рекурсивную функцию, в ней необходимо указать, в какой момент следует прервать рекурсию. **Каждая рекурсивная функция состоит из двух частей: базового случая и рекурсивного случая.** В рекурсивном случае функция вызывает сама себя. В базовом случае функция себя не вызывает, чтобы предотвратить заикливание.

Разветвленный (условный) алгоритм





```
const fact = (n) => {  
  if (n === 1) {  
    return 1;  
  } else {  
    return n * fact(n - 1);  
  }  
}
```

```
fact(5); // 120
```



```
const fact = (n) => {  
  let result = 1;  
  for (let i = n; i > 0; i--) {  
    result *= i;  
  }  
  return result;  
}
```

```
fact(5); // 120
```

f (4)



```
if (4 === 1) return 1;  
return 4 * f (4 - 1);
```

6



```
if (3 === 1) return 1;  
return 3 * f (3 - 1);
```

2



```
if (2 === 1) return 1;  
return 2 * f (2 - 1);
```

1



```
if (1 === 1) return 1;  
return 1 * f (1 - 1);
```