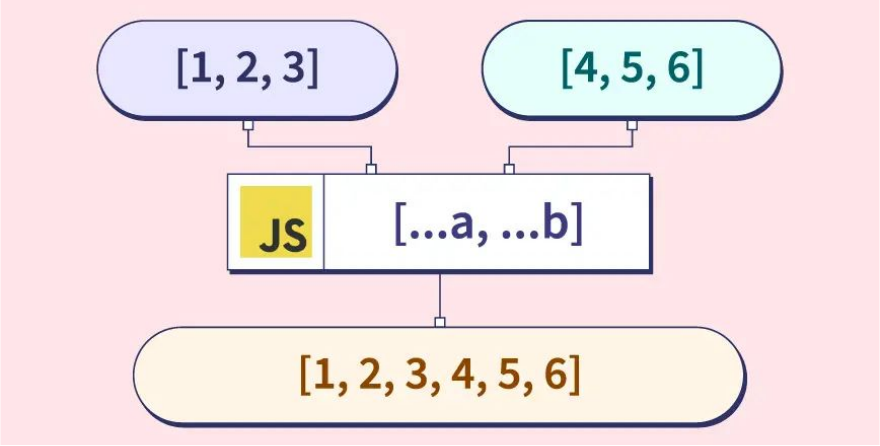
JavaScript

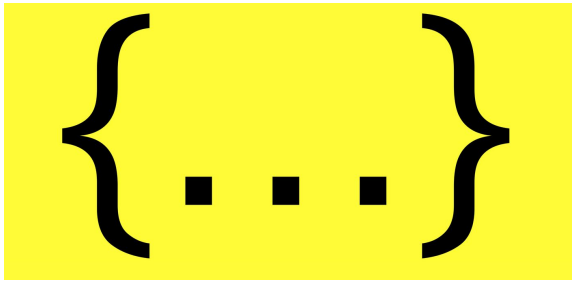
Destructuring assignments
Functions declarations, functional expressions, arrow functions

Деструктуризация



Оператор расширения - spread

В JavaScript оператор расширения (spread operator) представляет собой синтаксическую конструкцию, которая используется для создания копий массивов, объединения массивов, передачи аргументов функции и других подобных задач.



1. Для массивов:

В приведенном примере **...arr1** разворачивает элементы массива **arr1** в новом массиве **arr2**.

```
const arr1 = [1, 2, 3];
const arr2 = [...arr1, 4, 5, 6];
console.log(arr2); // [1, 2, 3, 4, 5, 6]
```

2. Для объектов:

Здесь **spread** оператор используется для создания нового объекта **obj2**, который содержит все свойства из **obj1**, а также новые свойства.

```
const obj1 = { key1: 'value1', key2: 'value2' };
const obj2 = { ...obj1, key3: 'value3', key4: 'value4' };
console.log(obj2);
// { key1: 'value1', key2: 'value2', key3: 'value3', key4: 'value4' }
```

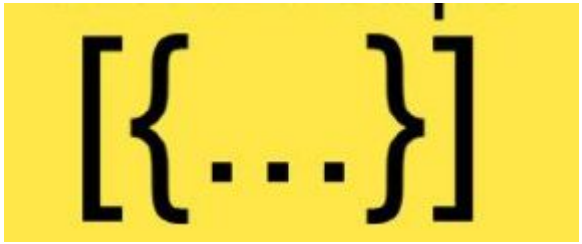
3. Передача аргументов функции

В данном примере spread оператор используется для передачи элементов массива в качестве аргументов функции **sum**.

```
function sum(a, b, c) {
  return a + b + c;
}

const numbers = [1, 2, 3];
console.log(sum(...numbers)); // 6
```

Деструктурирующее присваивание



Деструктурирующее присваивание – это специальный синтаксис, который позволяет нам «распаковать» массивы или объекты в несколько переменных, так как иногда они более удобны.

Деструктуризация массива

```
1  // у нас есть массив с именем и фамилией
2  let arr = ["Ilya", "Kantor"];
3
4  // деструктурирующее присваивание
5  // записывает firstName = arr[0]
6  // и surname = arr[1]
7  let [firstName, surname] = arr;
8
9  alert(firstName); // Ilya
10 alert(surname);  // Kantor
```

Деструктурирующее присваивание ничего не уничтожает, его задача – только скопировать нужные значения в переменные.

Деструктуризация объекта

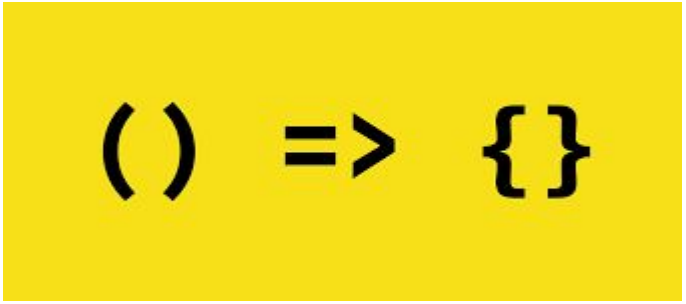
Деструктурирующее присваивание также работает с объектами.

```
let options = {
  title: "Menu",
  width: 100,
  height: 200
};

let {title, width, height} = options;

alert(title);  // Menu
alert(width);  // 100
alert(height); // 200
```

Functions



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 (изучили на предыдущих лекциях)

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  }
```

2. **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:

```
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

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

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) ); // 3
```

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); // 8

const differenceResult = operateOnNumbers(8, 3, subtract);
console.log(differenceResult); // 5
```

Передача анонимной функции в качестве параметра

Анонимные функции в JavaScript - это функции, которые не имеют имени и обычно определяются прямо в месте, где они используются.

```
// Функция, принимающая функцию в качестве параметра
function performOperation(x, y, operation) {
  return operation(x, y);
}

// Используем функцию performOperation с анонимной функцией-выражением
const result = performOperation(8, 3, function(a, b) {
  return a - b;
});

console.log(result); // Вывод: 5
```

Параметры по умолчанию

Если при вызове функции аргумент не был указан, то его значением становится undefined.

Если мы хотим задать параметру text значение по умолчанию, мы должны указать его после =:

```
1  function showMessage(from, text = "текст не добавлен") {
2    alert( from + ": " + text );
3  }
4
5  showMessage("Аня"); // Аня: текст не добавлен
```

Теперь, если параметр text не указан, его значением будет "текст не добавлен"

Альтернативные параметры по умолчанию

Иногда имеет смысл присваивать значения по умолчанию для параметров не в объявлении функции, а на более позднем этапе.

Во время выполнения функции мы можем проверить, передан ли параметр, сравнив его с **undefined**:

```
1  function showMessage(text) {
2    // ...
3    if (text === undefined) { // если параметр отсутствует
4      text = 'пустое сообщение';
5    }
6    alert(text);
7  }
8  showMessage(); // пустое сообщение
```