Функции первого порядка. IIFE. Замыкания. Контекст. Bind, call, apply.

Функции первого класса.

Функция - это сущность типа Объект. Функции - ведут себя как объекты. Мы можем записать функцию в переменную. Мы можем передать функцию в качестве аргумента в другую функцию. Мы можем вернуть функцию из другой функции.

В JavaScript мы имеем функции первого класса. Если язык программирования имеет функции первого класса, то значит они трактуются как объекты, то есть могут быть переданы другим функциям и их можно вернуть из функций. Так же их можно присваивать переменным.

функции в качестве *аргументов*:

```
function arrayCalc(arr, fn) {
  var newArray = [];
  for (var i = 0; i < arr.length; i++) {
    newArray.push(fn(arr[i]));
  }
  return newArray;
}

// Функция расчитывает возраст
function calcAge(item) {
  return 2020 - item;
}

// Рассчитаем возраст всех персон
var ages = arrayCalc(years, calcAge);
console.log(ages); // [30, 10, 35, 45, 8, 58]
```

Функции которые возвращают *функции:*

```
function interviewQuestion(job) {
  if (job === "дизайнер") {
    return function (name) {
      console.log(`Привет ${name}, расскажи что такое UX дизайн?`);
    };
  } else if (job === "учитель") {
    return function (name) {
      console.log(`Привет ${name}, расскажи какой предмет ты преподаешь?`);
    };
  } else {
    return function (name) {
      console.log(`Привет ${name}, расскажи чем ты занимаешься?`);
    };
  }
}

var designerQuestion = interviewQuestion("дизайнер");
designerQuestion("Петр");  //Привет Петр, расскажи что такое UX дизайн?
interviewQuestion("дизайнер")("Егор");  //Привет Егор, расскажи что такое UX дизайн?
```

IIFE - Анонимные самовызывающиеся функции

Их используют, чтобы ограничить область видимости переменных, сделать их приватными.

Пример: случайное число от 1 до 10, больше 5 – выиграл.

```
// 1. Global variant
var score = Math.random() * 10; // 1 - 10
console.log(score >= 5);
```

Переменную ***score*** здесь может изменить любой скрипт извне. Она не приватная

```
// 2. Function variant */
function game() {
```

```
  var score = Math.random() * 10; // 1 - 10
  console.log(score >= 5);
}
game();
```
Переменная ***score*** здесь скрыта, к ней нет доступа.

```
// 3. IIFE  variant
(function (name) {
  var score = Math.random() * 10; // 1 - 10
  console.log(score >= 5);
  console.log(name);
})("John");
```
При этом варианте не надо создавать новую функцию с именем, при этом функция сразу вызовется. При этом можно передать параметры.

аббревиатура IIFE - означает функцию, запускаемую сразу после объявления.

// Пути создания IIFE

```
(function() {
 alert("Скобки вокруг функции");
})();

(function() {
 alert("Скобки вокруг всего");
}());

!function() {
 alert("Выражение начинается с логического оператора NOT");
}();

+function() {
 alert("Выражение начинается с унарного плюса");
}();
```

Скобки вокруг функции – это трюк, который позволяет показать JavaScript, что функция была создана **в контексте другого выражения**, и, таким образом, это функциональное выражение: ей не нужно имя и её можно вызвать немедленно

! IIFE - это не повторно используемый код. Основное назначение - создать определенную область видимости с приватными переменными и сразу же вызвать этот код на выполнение.

Closures – замыкания

Замыкание: функция всегда имеет доступ к переменным и параметрам внешней функции, в которой она была определена. Даже когда внешняя функция выполнена и вернула значение.

Функция имеет доступ к переменным в области видимости, где она была определена. Даже если функция вызывается в другом месте. Даже если функция была определена в другой родительской функции и родительская функция была выполнена и вернула значения. Контекст выполнения уходит, но остается VO (variable object) все равно остается в памяти, и описанная внутренняя функция имеет доступ к нему. Scope Chain - это указатели на Variable Objects (переменные, доступные в области видимости функции).

```
function retirement(retirementAge) { // сколько лет до пенсии, аргумент – возраст выхода на пенсию
  var words = " лет осталось до пенсии";
  return function (yearOfBirth) {
    var age = 2020 - yearOfBirth; // 2020 - 1990 = 30
    var left = retirementAge - age; // 65 - 30 = 35
    console.log(left + words); // 35 лет осталось до пенсии
  };
}
var retirementRussia = retirement(65);
retirementRussia(1990); // 35 лет осталось до пенсии
var retirementUSA = retirement(66);
retirementUSA(1985); // 31 лет осталось до пенсии
```

Для внутренней функции ***retirementAge*** и ***words*** - внешние. Она использовала эти переменные в тот момент, когда **была определена**, но не когда вызвана.

Замыкание – это функция, которая запоминает свои внешние переменные (в момент, когда она была определена, с помощью скрытого свойства **[[Environment]])** и может получить к ним доступ. В JavaScript, все функции изначально являются замыканиями

Bind, call, apply - привязка контекста

```
var john = {
  name: "Джон",
  age: 28,
  job: "дизайнер",
  sayHello: function (style, timeOfTheDay) {
    if (style === "formal") {
      console.log(`Добрый ${timeOfTheDay}, дамы и господа. Меня зовут ${this.name}, моя профессия ${this.job}, мне ${this.age} лет.`);
    } else if (style === "friendly") {
      console.log(`Здарова! Как дела? Я ${this.name}, , я работаю ${this.job} и мне ${this.age} лет. Хорошего тебе ${timeOfTheDay}.` );
    }
  },
};

var mary = {
  name: "Мария",
  age: 30,
  job: "программист",
};
```

Обычное использование:

```
john.sayHello("formal", "утро"); // Добрый утро, дамы и господа. Меня зовут Джон, моя профессия дизайнер, мне 28 лет.
john.sayHello("friendly", "вечера"); // Здарова! Как дела? Я Джон, , я работаю дизайнер и мне 28 лет. Хорошего тебе вечера.
```

Метод call(). Позволяет вызвать метод объекта, но с привязкой контекста другого объекта. В первый аргумент call передается новый контекст для this, остальные аргументы передаются «как есть».

```
john.sayHello.call(mary, "formal", "утро"); // Добрый утро, дамы и господа. Меня зовут Мария, моя профессия программист, мне 30 лет.
john.sayHello.call(mary, "friendly", "дня"); // Здарова! Как дела? Я Мария, , я работаю программист и мне 30 лет. Хорошего тебе дня.
```

Метод apply() - аналогично методу call, но все аргумнеты передаются массивом.

```
john.sayHello.apply(mary, ["friendly", "вечера"]); // Здарова! Как дела? Я Мария, , я работаю программист и мне 30 лет. Хорошего тебе вечера.
```

Метод bind() - создает копию функции, с привязкой к новому контексту и устанавливает значения для аргументов (по умолчанию).

Запишем в переменную. Несмотря на то, что вызываем на jhon, в контексте его указываем.

```
var johnFriendly = john.sayHello.bind(john, "friendly", "вечера");
johnFriendly(); // Здарова! Как дела? Я Джон, , я работаю дизайнер и мне 28 лет. Хорошего тебе вечера.
```

Можем передать парамеры по-другому. Здесь мы передаем первый аргумент по умолчанию:

```
var maryFriendly = john.sayHello.bind(mary, "friendly");
```

А затем определяем второй аргумент

```
maryFriendly("вечера");  // Здарова! Как дела? Я Мария, , я работаю программист и мне 30 лет. Хорошего тебе вечера.

var maryFormal = john.sayHello.bind(mary, "formal");
maryFormal("вечер"); // Добрый вечер, дамы и господа. Меня зовут Мария, моя профессия программист, мне 30 лет.

console.dir(maryFormal);
ƒ bound sayHello()
length: 1
name: "bound sayHello"
arguments: (...)
```

```
caller: (...)
[[Prototype]]: ƒ ()
[[TargetFunction]]: ƒ (style, timeOfTheDay)
[[BoundThis]]: Object
        age: 30
        job: "программист"
        name: "Мария"
 [[BoundArgs]]: Array(1)
        0: "formal"
        length: 1
        [[Prototype]]: Array(0)
```

Можно также не записывать функцию в переменную, а сразу запустить на выполнение.

```
john.sayHello.bind(john, "friendly")("вечера");
```