STEP IT ACADEMY

Занятие №28

Прототипы. Контекст. Виды функций. Замыкания

Что такое область видимости переменных?

Переменная let

Ключевое слово let позволяет объявлять переменные с ограниченной областью видимости — только для блока {...}, в котором происходит объявление. Это называется блочной областью видимости.

Переменная const

Объявление констант с такой же видимостью, как и у let.

Это означает не то, что значение константы неизменно, а то, что идентификатор переменной не может быть пере присвоен.

```
{
    const ARR = [5, 6];
    ARR.push(7);
    console.log(ARR); // [5,6,7]
    ARR = 10; // TypeError
    ARR[0] = 3; // значение можно менять
    console.log(ARR); // [3,6,7]
}
```

Переменная var

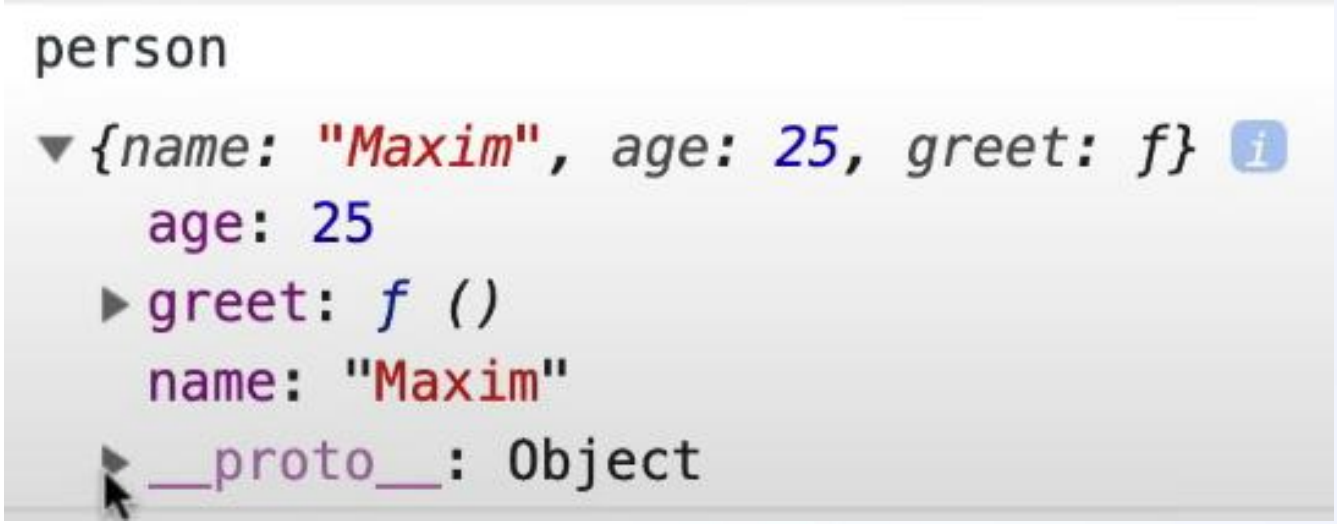
Переменная var можно создать как глобально так и локально с одинаковым названием. И обратиться к внешней переменной через конструкцию window. Такой способ не безопасный.

```
var x = 50;
function func(){
    var x = 10;
    console.log(x); //10
        console.log(window.x); //50
}
```

Также проблема в поднятии. Если мы попытаемся вызвать переменную let до её объявления у нас возникнет ошибка, что переменной ещё нет. Тогда как var скажет, что переменная undefined.

Прототипы

Это определённый объект (главный), который присутствует у объектов. И он вызывается по цепочке сверху вниз. Если мы находим какие-то поля или функции на верхним уровне, то мы обращаемся к ним. Если не находим их то идём вниз по прототипу и ищем там.



Контекст

Контекст напрямую завязан на использование функций с ключевым словом “this”. Так как функции всегда выполняются внутри какого-либо из объектов, контекст выполнения функции - это привязанный объект. Рассмотрим сразу два примера.

```
function hello() {
  console.log('Hello', this)
}
```

```
hello()
Hello                                   index.js:2
▶ Window {postMessage: ƒ, blur: ƒ, focus: ƒ
  , close: ƒ, parent: Window, …}
```

```
const person = {
  name: 'Vladilen',
  age: 25,
  sayHello: hello
}
person
▶ {name: "Vladilen", age: 25, sayHello: ƒ}
person.sayHello()
Hello                                   index.js:2
▶ {name: "Vladilen", age: 25, sayHello: ƒ}
```

Смена контекста

У функций есть несколько методов для этого:

- bind - который привязывает метод к какому-либо объекту: fn.bind(obj, args). Где obj - объект, к которому нужно привязаться, args - аргументы функции, через запятую.
- call - точно также как и bind но сразу вызывает функцию.
- apply - работает схожим образом с bind, только аргументы функции записываются не через запятую, а в массив: fn.apply(obj, [args])

Переменная globalThis

globalThis - нововведение, статичная переменная, постоянная ссылка на глобальный объект окружения.

Виды функций

Function Declaration – они создаются интерпретатором. То есть можно вызвать до объявления функции.

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

Function Expression – вызываются строго после объявления функции.

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

Named Function Expression – можно внутри функции вызывать саму себя. Например для рекурсии. Факториал!

```
const funk = function sum (a, b) {
  return a + b;
}
```

Стрелочные функции

Представляют собой сокращённую запись функций в ES6. Стрелочная функция состоит из списка параметров (...), за которым следует знак => и тело функции.

```
let mul = (a, b)=> {return a * b}
```

```
btn.onclick = function(){
    console.log(this)
}
```
//this в данном случае будет ссылкой на кнопку

Если аргументов функции нет, то можно оставить скобки пустыми.

Главное и важное отличие стрелочных функций и обычных функций - в контексте выполнения и ссылки на объект this. Стрелочные функции не имеют своего контекста, в отличии от остальных функций

```
btn.onclick = ()=>{
    console.log(this)
}
```
//this в данном случае будет ссылкой на глобальный объект window

Замыкания

По сути это функция внутри другой функции.
Рассмотрим пример функции внутри другой функции.

На первом рисунке ничего не выведет. Потому что первая функция возвращает новую функцию. Поэтому можно занести результат в переменную и вызвать эту переменную как функцию.

Когда вызывали 1 функцию и передали туда число 42. Она отработала и вернула новую функцию. А так как 2 функцию в контексте 1 функции, то переменная N как бы замкнута там. Значение N уже известно для 2 функции.

```
function createCalcFunction(n) {
  return function() {
    console.log(1000 * n)
  }
}

createCalcFunction(42)
```

```
function createCalcFunction(n) {
  return function() {
    console.log(1000 * n)
  }
}

const calc = createCalcFunction(42)
calc()
```

Давайте подведем итоги!

Чему мы научились?

Что мы использовали?