# Синтаксис языка сценариев

# 1. JavaScript - основные понятия

Изначально ***JavaScript*** был создан, чтобы «сделать веб-страницы живыми».
Программы на языке ***JS*** называются ***скриптами***.

Сегодня ***JS*** может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся ***«движком» JS***.

***«Движок» JS*** — специализированная программа (*интерпретатор*), обрабатывающая текст на ***JS***.

Этапы работы «***движка»***.

1. Движок читает («*парсит*») текст скрипта.
2. Затем он преобразует («*компилирует*») скрипт в машинный код.
3. После этого машинный код исполняется («интерпретируется») и работает достаточно быстро.

«Движок» может применять оптимизации на каждом этапе.

Спецификация ***ECMA-262*** содержит самую глубокую, детальную и формализованную информацию о ***JavaScript***. Она определяет сам язык.

# 2. JavaScript - основные понятия.

***Инструкции*** – это синтаксические конструкции и команды, выполняющие действия.

Инструкции ***могут*** отделяться точкой с запятой (***«;»***).

В большинстве случаев ***«;»*** можно не ставить, если есть переход на новую строку. НО !!!

```
Val1=3; Val2=4
Val3=5
```

```
Val1=3
Val2=4
Val3=5
```

```
Val1=3;
Val2=4;
Val3=5;
```

```
Val1=3 Val2=4
Val3=5;
```

***Рекомендуется*** ставить ***«;»*** между инструкциями, даже если они отделены переносами строк.

# 3. Ключевые  и зарезервированные слова

**Ключевые и зарезервированные слова нельзя использовать в качестве идентификаторов.**

| break | case | catch | const |
|---|---|---|---|
| continue | default | do | else |
| finally | for | function | if |
| in | new | prototype | return |
| switch | this | typeof | throw |
| try | var | while | with |
| boolean | short | interface | implements |
| double | yield | private | long |
| extends | import | float | byte |
| static | protected | native | int |
| enum | char | super | public |
| package | goto | export | class |
|  |  | throw |  |

# 4. Комментарии

***JS*** предоставляет комментарии для одной строки (или даже ее части со знака комментария и до конца строки), а также и для целой серии строк:

```
// одна строка

/*
много строк
*/
```

**Вложенные многострочные комментарии не поддерживаются!**

# 5. Переменные

Регистр имеет значение !

Для создания переменной в ***JS*** используйте ключевое слово ***let***.

Определить переменную значит поместить в нее данные используя оператор присваивания (=)

Можно объявить несколько переменных в одной строке:

```
let message;
message = 'Hello';
```

Нелатинские буквы разрешены, но не рекомендуются !

```
let user = 'John', age = 25, message = 'Hello';
```

```
let user = 'John',
    age = 25,
    msg = 'Hello';
```

!

```
let user = 'John';
let age = 25;
let msg = 'Hello';
```

Ключевое слово ***var*** – почти то же самое, что и ***let***. Оно объявляет переменную, но немного по-другому, «устаревшим» способом. Объявить неизменяемую переменную, можно используя ***const*** вместо ***let.***

*Создание переменной без режима «use strict»:

```
num = 5;        // если переменная "num" раньше не
                //существовала, она создаётся
alert ( num ); // 5
```

Переменная в ***JS*** может хранить любые данные. В один момент там может быть строка, а в другой – число!

```
let user = ‘John’,
    user = 1234,
```

# 6. Переменные

**Тип переменной задается при ее инициализации значением**. Если переменную просто объявить и не инициализировать значением, то ее значение будет ***undefined***, а тип - ***Undefined***.

```
let x;            // Переменная без инициализации
let y = 100;      // Переменная с инициализацией
let i, j = 100; // Определение нескольких переменных
                  // с инициализацией одной из них
```

В ***JS*** есть ограничения, касающиеся имён переменных:

I. Имя переменной должно содержать только буквы, цифры или символы $ и _.
II. Первый символ не должен быть цифрой.
*III.* *Не должно содержать* никаких спец. символов (!, ?, I, <,…);
*IV.* *Не должно совпадать* с **ключевыми словами;**
V. Одно и то же имя нельзя определять в пределах одной области видимости более 1 раза.

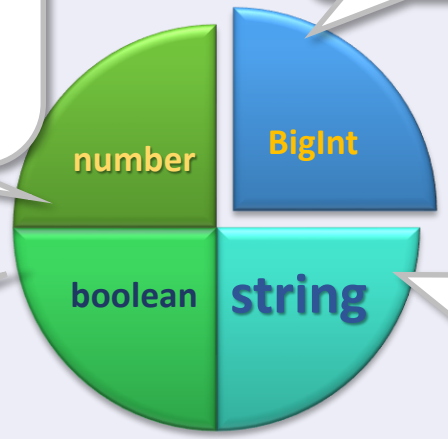
# 7. Предопределенные типы данных

В ***JS*** не нужно задавать тип данных при объявлении, так как он определяется автоматически при присваивании или определении и сохраняется до тех пор, пока не будет выполнено следующее присваивание.

64-битное знаковое число в стандартном или экспоненциальном формате:
**var fExp 23.4524E-12;**
**var f = -13.3451;**
**var xDec = 123;**
**var xHex = 0xFF;**
От -((2^53)-1) до ((2^53)-1)
***Infinity, -Infinity, NaN***

целые числа произвольной длины

```
// символ "n" в конце означает, что это BigInt
const bigInt = 1234567890123456789012345678901234567890n;
```




Ряд символов в формате UNICODE в ‘ ‘ , “ “ или ` `:
**var str1 = "Hello";**
**var str2 = 'Hello’;**
**var str3 = "80's";**
**var str4 = ` Hello`;**

Принимает одно из 2-х значений (**true, false**):
**var b = true;**
**b = (3 == 5);** //=> b = false

```
let isGreater = 4 > 1;
alert( isGreater ); // true (результатом
                    // сравнения будет "да")
```

*В переменных строкового типа можно использовать специальные символы:

| Символ | Значение |
|---|---|
| \b | Возврат на символ с его удалением |
| \t | Горизонт. Табуляция |
| \n | Новая строка |
| \v | Вертикальная табуляция |
| \r | Возврат каретки |
| \” | Двойные кавычки |
| \’ | Одинарные кавычки |
| \\ | Обратная косая черта |

```
let name = "Иван";
// Вставим переменную
alert( `Привет, ${name}!` ); // Привет, Иван!

// Вставим выражение
alert( `результат: ${1 + 2}` ); // результат: 3
```

Выражение внутри ${…} вычисляется, и его результат становится частью строки.

# 8. Предопределенные типы данных

Спец. значение ***null*** не относится ни к одному из типов, описанных выше. Оно формирует отдельный тип, который содержит только значение *null*:

```
let age = null;
```

*Это просто специальное значение, которое представляет собой «ничего», «пусто» или «значение неизвестно».

```
let age;
alert(age); // выведет "undefined"
```

Спец. значение ***undefined*** формирует тип из самого себя так же, как и ***null***.

```
let age = 123;
// изменвм  на undefined
age = undefined;
alert(age); // "undefined"
```

Оно означает, что «значение не было присвоено».

Тип ***object*** (объект) – особенный.

В объектах хранят коллекции данных или более сложные структуры.

Оператор ***typeof*** возвращает тип аргумента.

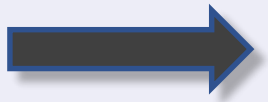
```
typeof 5 // Выведет "number“

  // Синтаксис, напоминающий вызов
  // функции (встречается реже)
typeof(5) // Также выведет "number"
```

```
typeof 50 + " Квартир"; // Выведет "number Квартир"
typeof (50 + " Квартир"); // Выведет "string"
```

# 9. Преобразования типов

```
let n = 7;
let f = 5.2018;
let b = true;
let str =  "Hello";
let unknown;
let bn = 10n;
```



```
alert(typeof(n));    //=> number
alert(typeof(f));    //=> number
alert(typeof(b));    //=> boolean
alert(typeof(str));  //=> string
alert(typeof(unknown)); //=> undefined
alert(typeof(Math)); // => object
```

Тип любой переменной можно узнать в процессе выполнения сценария при помощи оператора **typeof** ( )

```
let n = 56;       // Целое значение
let f = 0.27;     // Вещ. значение
let res = n * f; // Результат  - вещ.: 15.12
                  // неявное преоб-е
```

VS

```
let f = 3.5;
let n = parseInt(f);
// 3.5 -> ”3.5” -> 3
alert(n); //=> 3
```

**parseInt**(f);
**parseFloat**(n)
(явное преобразование)

*Тип переменной можно изменить в любой момент времени, присвоив переменной значение другого типа. А так же использовав явные или неявные преобразования типов:

```
let n = 7, f = 5.2018, b = true;
let res1 = n + " is a number";
let res2 = f  + " is a float number";
let res3 = b + " is a boolean value";
```

```
alert(res1);  //=> “7 is a number”
alert(res2);  //=> “5.2018 is a float number”
alert (res3); //=> “true is a boolean value”
```

Здесь ! Все числа преобразуются в строку.

# 10. Практика

```
val = Number(str_);
```

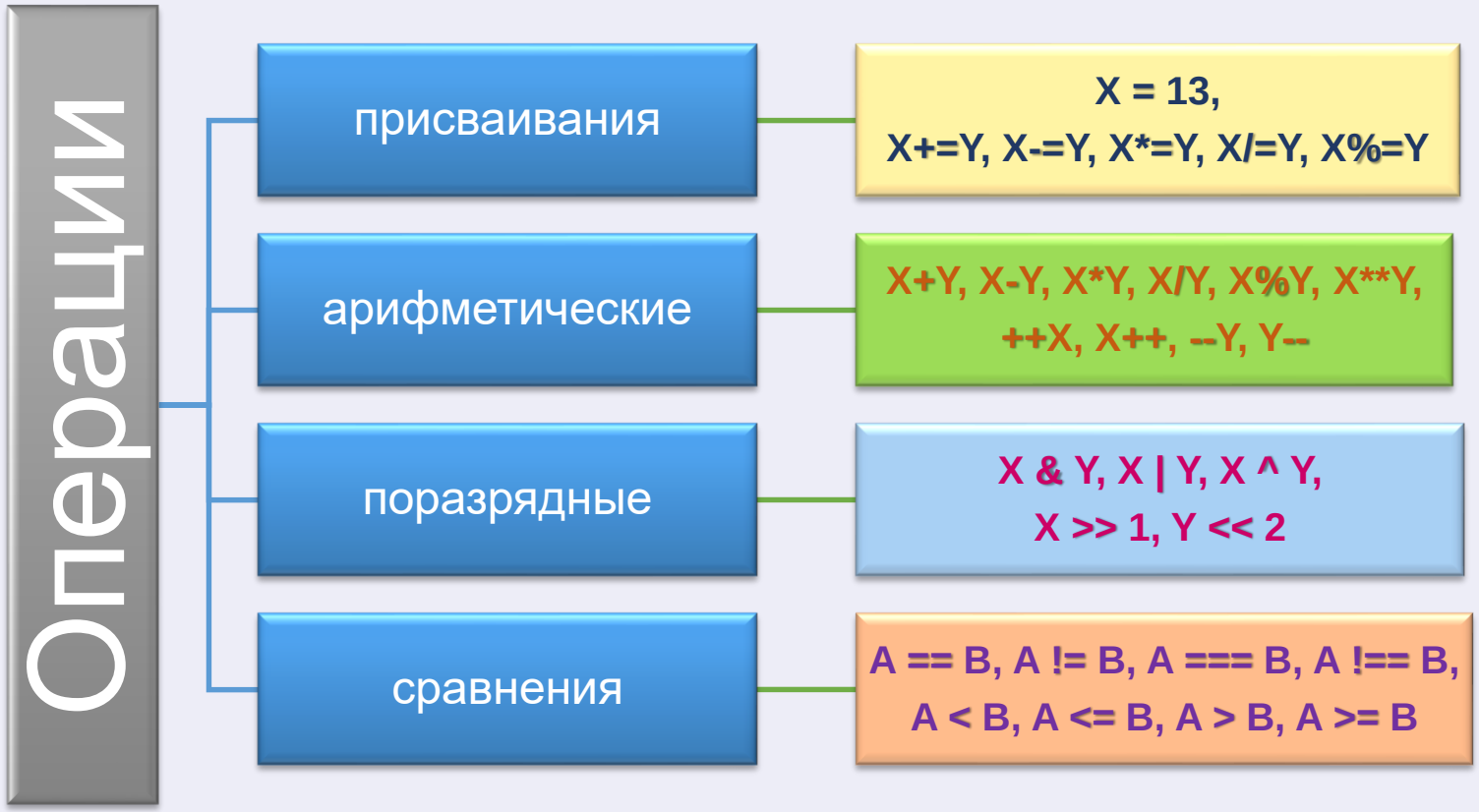
```
str = String(val_);
```

```
bool = Boolean(str_);
str_ : [“Hello”, “0”, “ ”, “” ]
```

```
bool = Boolean(val_);
val_: [0,1,null, NaN, undefined]
```

? Выясните результаты преобразований на практике.

# 11. Операции




| № | Приоритет |
|---|---|
| | (n), a[i], new T() |
| | +n, -n,~n, !b, i++,i--, !b, ++i,-- i, typeof () |
| | n*n, n/n, n%n, n+n, n-n, n<<i, n>>i, n>>>i |
| | n<n, n<=n, n>n, n>=n, obj instanceof T, str in obj |
| | n==n, n!=n, n===n, n!==n |
| | n&n, |
| | n^n, |
| | n\|n, |
| | b&&b |
| | b\|\|b |
| | b?x:x |
| низкий | n=n, n+=n, n-=n, n*=n, n/=n, n%=n, n<<=n, n>>=n, n>>>=n, n^=n, n&=n, n\|=n |

«**Проверка на идентичность (===)**». При выполнении данной операции неявное преобразование типов не выполняется.

!

«**Проверка на равенство (==)**». При выполнении данной операции производится неявное преобразование к типу первого операнда.

# 12. Управляющие структуры

По средствам управляющих структур в *JS* можно управлять ходом выполнения ПО.

# 13. Исключительные ситуации

```
try {
    // сомнительные действия, может
    // возникнуть ошибка
    [throw error];
}
catch (error) {
     // действия обработки ошибки,
    // если ошибка возникла
}
final {
     // завершающие действия,
    // выполняются вне зависимости
    // возникла ошибка или нет
}
```

```
let file = new File("file.dat");
try {
     file.open(File.ReadOnly);
}
catch(e) {
 alert("Code error:" + e);
}
finally {
    file.close();
}
```

```
try {
    Lalala;
}
catch(e) {
   alert("Code error:" + e.name);
   alert("Code error:" + e.message);
   alert("Code error:" + e.stack);
// alert("Code error:" + e);
}
```

```
let  res =0;
try {
     if(res==0){
        throw new Error("res, must > 0");
      }
   alert(res);
}
catch(e) {
  alert("Code error:" + e);
}
```

При помощи оператора ***throw*** можно генерировать свои собственные исключения для их последующего перехвата в операторе ***try … catch***. Исключение может быть строкой, числом или объектом класса. Выброс исключения в данном случае выглядит : ***throw error***;

# 14. Функции

***Функция*** – блок действий над полученными данными с последующим возвращением результата. Объявление осуществляется при помощи ключевого слова ***function***. Функции можно определять в любом месте программы.

```
function имя ([аргумент1], [аргументN])
{
    [действия]
}
```

Переданные в функцию значения всегда **копируются в локальные переменные** и используются в теле функции

Имя

Нет значения по умолчанию (===undefined)

Есть значение по умолчанию

```
function multiply(var1, var2 = 3) {
    return var1 * var2;
}
//      return
//       var1* var2;
```

Определение

```
var f = multiply(2.3, 13.7);
```

Вызов

```
var myMultiplyFunction = function [multiply]( var1, var2){
                                    return (
                                       var1 + 2
                                     * var2)
                                  };
var f = myMultiplyFunction( 2.3, 13.7);
```

Ключевое слово ***function*** позволяет создавать функции «На лету», как только в этом появится необходимость:

# 15. Функции

```
let myMultiplyFunction = new Function (“var1” , “var2” ,
                                        “return    var1 * var2;”  ) ;
let f = myMultiplyFunction(2.3, 13.7);
```

*При помощи специального объекта ***Function*** можно создавать и изменять функции в процессе выполнения самой программы. Благодаря этому можно обеспечить пользователю возможность задавать действия функции самому, набрав ее в текстовом поле программы.

Кол-во и значения аргументов, переданных функции, можно получить в самой функции при помощи встроенной в язык переменной ***arguments***. Эта переменная называется ***объектом активизации функции*** и инициализируется всякий раз при вызове функции. Т. о. можно определить функцию с любым кол-вом переданных в нее значений:

```
function multiply () {
    let res = 1;
    for (let i = 0; i < arguments.length; ++i) {
          res *= arguments[i];
   }
  return res;
}
```

```
var f = multiply(34.5, 14.2, 8.7, 3.4, 7.1);
```

Функцию «multiply» можно вызвать с любым кол-вом параметров

# 16. Рекурсивный вызов функции и объявление переменной

```
function factorial (n) {

if ( (n == 0) || (n == 1)) {
        return 1;
   }
   else {
           result = (n * factorial(n - 1));
   }
   return result;
}
//-----------------------------------------
alert("Factorial 10 = " + factorial(10));

//=>"Factorial_10 = 3628800"
```

Условие завершения

Рекурсивный вызов

Вызов

Если внутри функции объявить переменную посредством **let,** то она будет являться ***локальной переменной***, то есть по завершению работы функции будет разрушена. Но если просто использовать внешнюю переменную - она ***глобальная*** и после завершения работы функции продолжит существовать.

```
let m = 'Hello';
function foo () {
     m = 2;
    //  let m=2;
}
foo();
alert (m) ; //=>2
```

Это глобальная переменная

Это была бы локальная переменная

# 17. Встроенные функции

***JS*** предоставляет ряд функций, определенных в глобальном объекте и являющихся частью самого языка.

**eval ()** –
выполняет содержимое строки, понимаемое как код ***JS***. Это может использоваться, например, для того, чтобы пользователь в процессе работы самой программы сценария вводил целые программные фрагменты на ***JS*** для их последующего выполнения.
Например:

```
let code = “ let i=0; ++i “
let result = eval (code );
```

**isNaN ()** –
возвращает значение *true*, если переданное выражение не является числовым значением;

**parseInt ()** –
преобразует переданную строку к целому типу. В случае неудачи функция возвращает значение *NaN*.

**parseFloat ()** –
преобразует переданную строку к вещественному типу. В случае неудачи функция возвращает значение *NaN*;

# 18. Домашка #3

1. Создать Qt-приложение «***StudyJS***» с помошью которого можно изучать синтаксис *JS* (работу операторов, проверять работу функций преобразования переменных, изучать работу функций).

*За основу можно взять проект «QtScriptBaseDemo» из «Практики» Занятия #2 “Введение в скрипты”