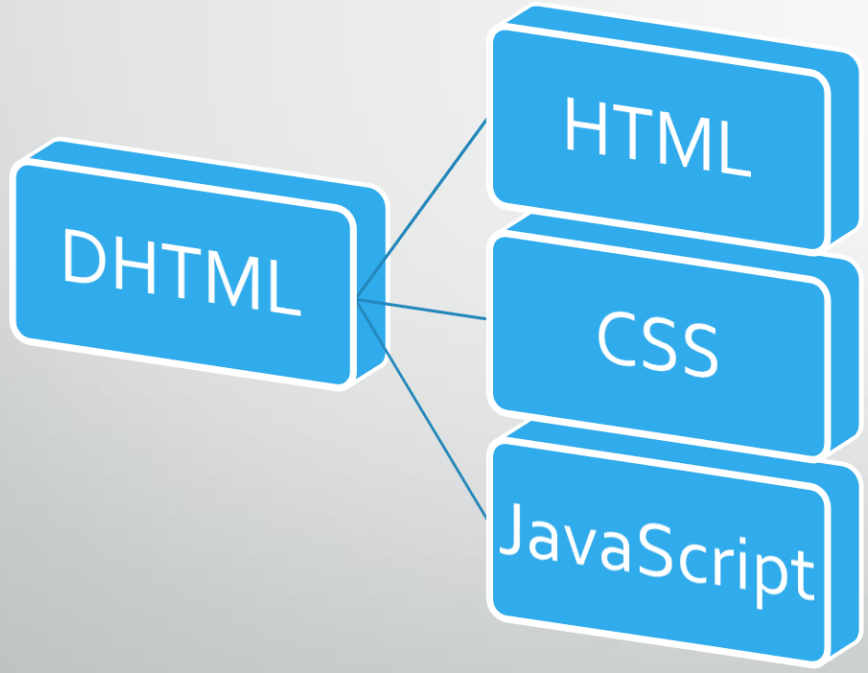
IT-Academy.BG

Основи на програмирането с JavaScript

Димитър Митев

Динамичен HTML (DHTML)



- Динамичен HTML(DHTML)
  - Съвкупност от съдържание, стилове и програмна логика, които правят възможно, страницата да реагира на промените и потребителски действия бързо и адекватно
- **HTML** -> *съдържанието на страницата*
- **CSS** -> *начина, по който изглежда страницата*
- **JavaScript** -> *логика за връзка между потребителя и системата*

IT-Academy.BG

JavaScript - предимства

- Валидация на клиентски данни
- Промяна на потребителския интерфейс и облик
- Динамична промяна на съдържанието
- Сложни изчисления
- Кустомизирани HTML контроли

JavaScript - възможности

- Валидация на клиентски данни
- Промяна на потребителския интерфейс и облик
- Динамична промяна на съдържанието
- Асинхронност
- Обработка на грешки
- Промяна на данни, съхранявани от браузъра

JavaScript - engines

- Зависят от браузъра
  - Google Chrome -> *V8*
  - Internet Explorer/Edge -> *Chakra*
  - Firefox -> *Spider Monkey*
  - Safari -> *JavaScriptCore*
  - *И др.*

Използване на JavaScript

- JavaScript кодът може да използван по следните начини:
  - <script> таг в *head* елемента
  - <script> таг в *body* елемента -> не се препоръчва
  - Външни файлове, реферирани чрез <script> елемент
    - **.js** файлове

```
<script src="scripts.js" type="text/javscript">
        <!-- code placed here will not be executed! -->
</script>
```

Изпълнение на JavaScript

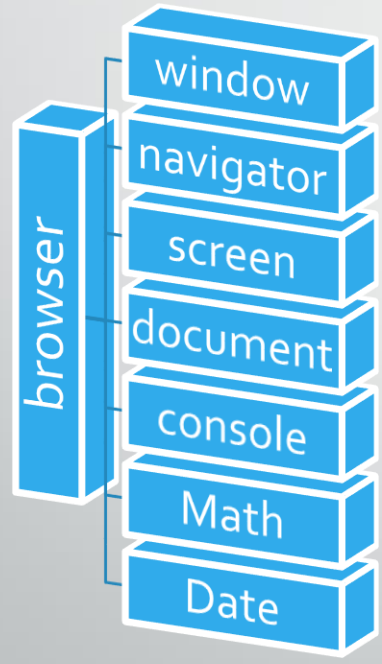
- JavaScript се изпълнява, докато се зарежда страницата или когато в браузъра е възникнало събитие (event was fired)
  - **Няма компилация. Няма проверка при компилация**
  - Целият код се изпълнява по време на зареждането на страницата
  - *Някои части от кода, могат да бъдат закачени за събития (events), които да се изпълнят при тяхното настъпване (fired event)

IT-Academy.BG

JavaScript синтаксис

- Оператори (+,-,*,/, =, != ….)
- Променливи (JS **не** е типизиран)
- Условни конструкции (if, else, switch)
- Цикли (for, while, forin)
- Масиви и асоциативни масиви
- Функции и функции променливи

Вградени обекти в браузъра

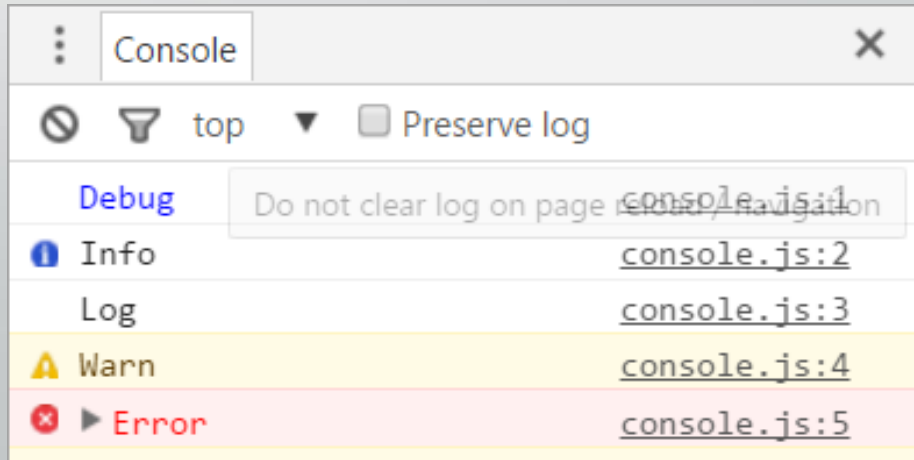


- Браузърът предоставя обекти на готово
  - **Window** -> *съдържа информация за текущия прозорец (tab) на браузъра*
  - **Navigator ->** *съдържа информация за местоположение и т.н.*
  - ***Document*** -> *съдържа информация за заредения документ (HTML)*
  - **Screen** -> *съдържа информация за потребителския екран (размери, цветова схема...)*
  - **Console** -> *позволява да се изписва инфорамция по време на изпълнението на кода (*заб.: необходимо е да има наличен инструмент за дебъгване)*
  - **Math** -> *съдържа методи, улесняващи мат. операции*
  - **Date** -> *дава възможност за работа с дати/календари*
- Повече инфромация

IT-Academy.BG

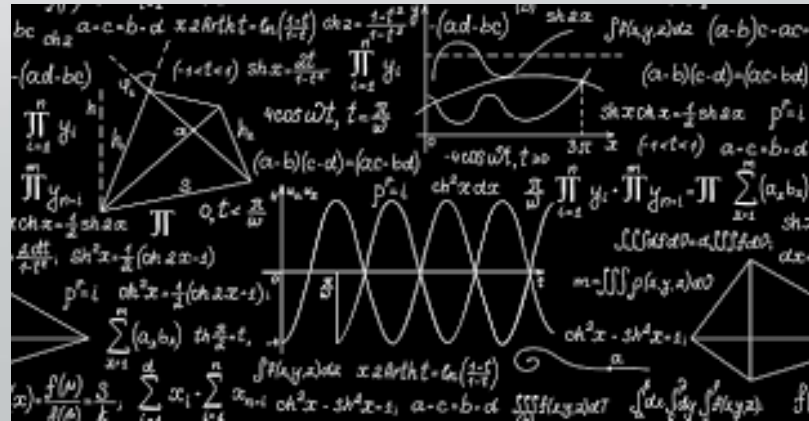
Обектът **console**

- Обектът **console** може да бъде използван само, ако е възможно дебъгването (т.е. имаме *debugging tool*). Използва се, за да се проследява информация (данни) по време на изпълнението на кода (*logging*)
- Обектът **console** има следните методи
  - *debug(message) -> *не бива да се използва, тъй като се счита за остарял способ*
  - *info(message)*
  - *log(message)*
  - *warn(message)*
  - *error(message)*
- *Прочетете повече за console*



Обектът **Math**

- Обектът **Math** дава достъп до различни видове математически операции
- Неговите property-та и методи се достъпват чрез точкова нотация (.)
  - *Math.PI* -> числото π
  - *Math.E* -> Неперово число
  - *Math.random()* -> дава произволно число
  - *Math.pow(base, exponent)* -> повдига число на степен
  - *Math.floor(x)* -> връща максималното число ≤ от даденото
  - Повече информация за обектът **Math**

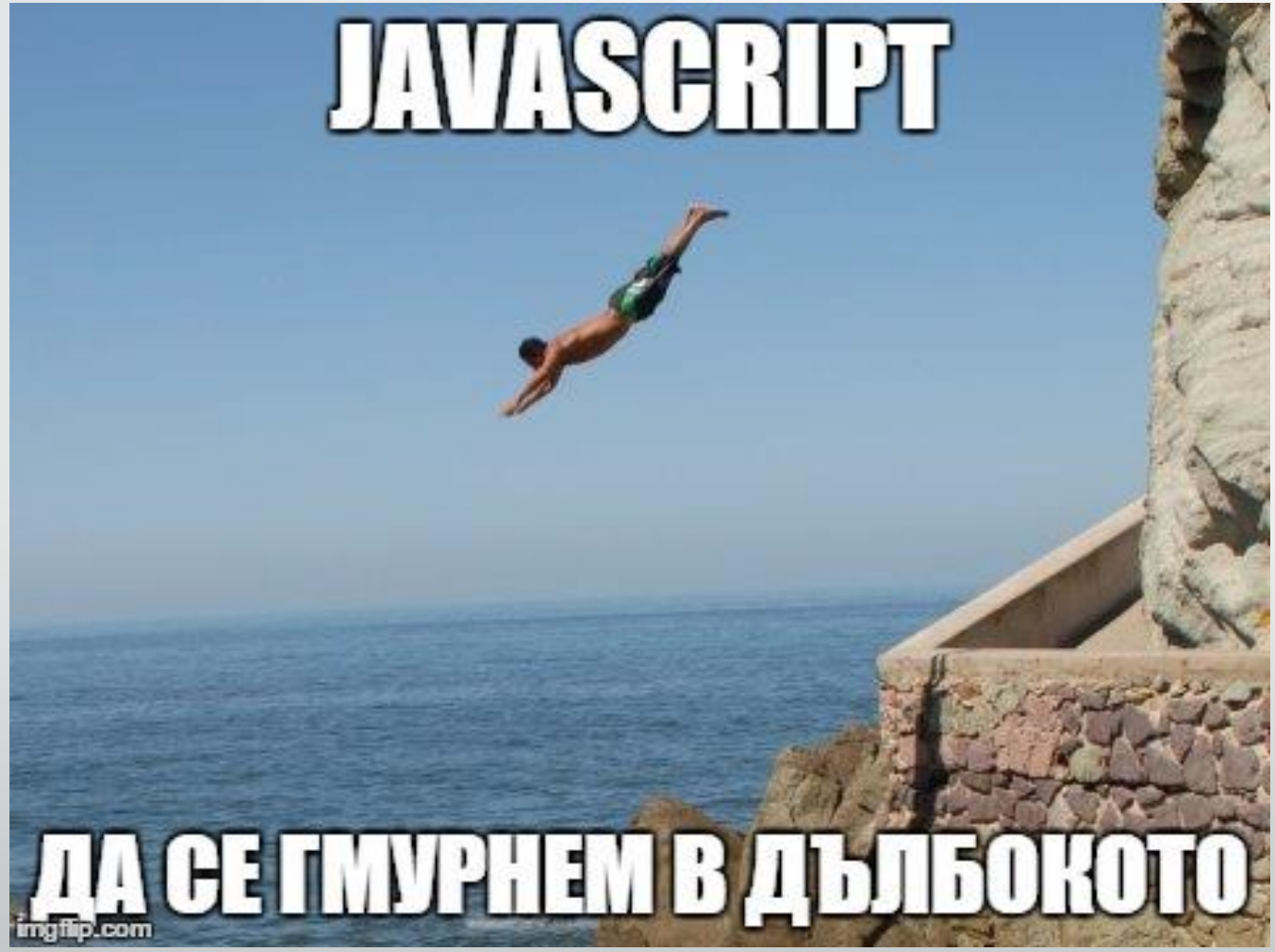


IT-Academy.BG

Таймери и интервали

- *setTimeout(callback, interval)* -> функциата (*callback*) ще се изпълни веднъж след определено време (*interval*), нарича се таймер.
- *clearTimeout(timer)* -> прекратява таймера
- setInterval(func, interval) -> функцията се ипълнява периодично през даден интервал от време
- *clearInterval(timer)* -> прекратява таймера
- Примери от W3School

Вградени обекти в браузъра


JAVASCRIPT
ДА СЕ ГМУРНЕМ В ДЪЛБОКОТО
imgflip.com

Типове данни и променливи.

- Данните се съхраняват като променливи в комп. памет
- Променливите обикновено се **характеризират** чрез своето **име, тип и стойност**
- **Тип на променлива (data type)** -> съвкупност от стойности с еднакви характеристики (*напр. цели числа /integers/*). Определя конкретния тип информация, който се съхранява в дадена променлива (*variable*)
- **JavaScript не е строго типизиран език**
  - **Всички променливи (*variables*) се декларират с ключовата дума var/let**
    - ***ключовата дума let е част от ES2015 и не се поддържа от по-стари версии на браузърите**
  - **Типа на променливата може да бъде сменян по всяко време**
  - **Все пак JavaScript има няколко основни типа от данни**

```
var radius = 5;
var name = 'Pesho';
let side  = 3.5;
```

Целочислен тип (Integer)

- Целочисления тип от данни представя цели числа (*1,2,3…55….1034…*)
- Диапазонът от стойности е от *-9007199254740992* до *9007199254740992*
- Поведението и начина на запазване зависи от конкретната имплементация на JS интерпретатора (*engine-а на browser-а*)
- Текстово представяне на число може да бъде пръврнато в целочислено число чрез вградената функция *parseInt(number)*

```
var count= 5;
let sum = 0;
var radius = parseInt(“5”);
```

Реални числа (floating point numbers)

- Реалните числа имат определена точност
- Могат да се появят аномалии при изчисления и трябва да се внимава
- Диапазонът им зависи от това колко битова ОС и браузър използваме (*32bit vs 64bit*). Все пак това могат да бъдат числа от *5e-324* до *1.79e+308*
- Базирани са на стандарта IEEE-754
- Символен низ може да се конверира към реално число чрез *parseFloat(x)*

```
var PI = Math.PI; // 3.141592653589793
var minValue = Number.MIN_VALUE; // 5e-324
var maxValue = Number.MAX_VALUE; // 1.79e+308
```

Аномалии при реалните числа

- При сравнението на реални числа може да се получи аномалия и трябва да се внимава

```
var a = 0.1;
var b = 0.2;
var sum = 0.3;
var equal = (a+b == sum); // false!!!

console.log(equal);
```

Числа. Обобщение

- Числовите типове в JS могат да се обощят чрез обекта **Number**
- Всички числа са реални, като могат да се превърнат в цели
- Трябва да се внимава дали се работи със символни низове или числа
- Хитрости

```
var valueDouble = 8.75;
var valueInt = valueDouble | 0; // 8

var roundedInt = (valueDouble + 0.5) | 0; // 9

var str = '1234';
var i = str | 0 + 1; // 1235
```

Булев тип (Boolean)

- Булевият тип данни има само 2 възможни стойности
  - **True**
  - **False**
- Използва се при логически изрази и условни конструкции

```
var a = 1;
var b = 2;

var isAGreaterThenB = (a > b);
console.log(isAGreaterThenB ); // false

var isEqualTo1 = (a == 1);
console.log(isEqualTo1); // true
```

Масив (Array)

- Масивът е съвкупност от елементи, ([1,2,3], [‘Pesho’, ‘Gosho’]…)
- Елементите имат фиксиран ред
- Масивите в JS имат дължина, но тя не е фиксирана
- Масивите се декларират чрез литерал *[ ]* или чрез констуктор *new Array(…)*
- Масивите могат да бъдат достъпвани по индекс
  - Започва от 0
  - Инкрементира се 1
  - Достига до дължината на масива - 1

```
var numbers = [1, 2, 3, 4, 5];
var mixed = [1, new Date(), 'hello'];
var matrix = [
            ['0,0', '0,1', '0,2'],
            ['1,0', '1,1', '1,2'],
            ['2,0', '2,1', '2,2']
];
```

Символни низове (String)

- Представя съвкупоност (низ) от символи
- Символните низове използват Unicode
- Символните низове са заградени в кавички (‘ ‘ или “ ”). Предпоръчва се да се изполват ед. кавички (‘ ‘)
- Символните низове могат да бъдат удължавани/съставяни като се използва оператора **плюс (+)**
- С последната версия на JS се въвежда стрингова интерполация, но тя работи само на последните версии на браузърите.
- Символните низове могат да се обхождат чрез индексите на техните символи и имат подобно поведение на масивите.

```
var greeting = ‘Hello World!’;
var niceGreeting = greeting + “It’s a nice day”;

console.log(niceGreeting);
```

```
var name= ‘Пешо’;
var surname = ‘Пешев‘;

console.log(`Привет, ${name} ${surname}`);
```

Undefined & null

- В JS съществува специален вид променлива *undefined*, който на практика значи,че промениливата не е инициализирана
- Също така съществува и тип *null*, който пък означава че променливата няма присвоена стойност
- *undefined* е различно от *null* и не бива да бъдат бъркани

```
var x;
alert(x); // undefined

x = null;
alert(x); // null
```

Проверка на типа на променлива

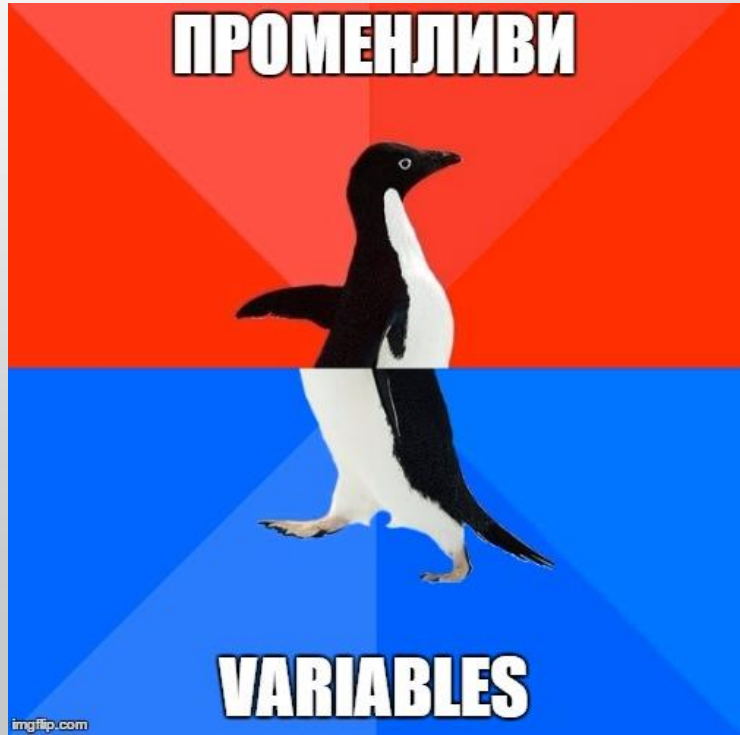
- Типът на промеливата може да бъде проверен по време на изпълнението на кода чрез *typeof(variable)*

```
var x = 5;
console.log(typeof(x)); // number
console.log(x); // 5

x = new Number(5);
console.log(typeof(x)); // object
console.log(x); // Number {}

x = null;
console.log(typeof(x)); // object
x = undefined;
console.log(typeof(x)); // undefined
```

IT-Academy.BG

Променливи

- Служи като контайнер за информация, който може да бъде променян по време на изпълнение на кода.
- Парче от комп. памет
- Променливите позволяват
  - Съхранението на информация
  - Манипулирането на съхр. Информация
- Всяка промелива се характеризира с
  - Име
  - Тип
  - Стойност



Деклариране на променливи

- Използва се ключовата дума *var/let*
- Когато се декларира променлива:
  - Трябва да има **подходящо** име
  - Може да се зададе първоначална стойност
  - Типът се определя от зададената стойност
- Синтаксис -> *var <име> [= <начална стойност>];*
- Пример -> *var radius = 5,4;*

Именуване на променливи

- Имената на променливите могат да съдържат
  - Букви (Unicode)
  - Цифри
  - Символи (_, $)
- Имената могат да започват **само** с буква или долна черта (_)
- Имената **не могат** да бъдат **ключови думи** в JavaScript
- Прието е променливите да се изписват в camelCase

Именуване на променливи(2)

- Имената на променливите трябва да бъдат описателни
- Препоръчва се използването на латински букви и имена на английски
- Не бива да бъдат нито прекалено къси, нито прекалено дълги
- Имената в JS са чувствителни към малки и главни букви (*case-sensitive*),т.е. променливата *Name* е различно от променливата *name*

IT-Academy.BG

Примери за правилно и неправилно именувани променливи

- Правилно именувани променливи

```
var radius = 5;
var greeting= 'Hello';
var newValue = 15;
var toAchieve= 100;
var privateClientsCount= 100;
```

- Неправилно именувани променливи

```
var Radius = 5;
var поздрав = 'Hello';
var new = 15;
var 2Achieve= 100;
var numberOfPrivateClientOfTheFirm = 100;
```

Присвояване на стойност на променлива

- Присвояването на стойност се извършва чрез оператора равно *(=)*
- Присвояването може да бъде каскадно. Операцията присвояване е дясно асоциативна,т.е. действията се изпълняват от дясно на ляво (напр. *var sum = 5 + 4;*)

```
var radius = 5;
var sideA = sideB = 12;
var area = sideA * sideB;

var first = 3;
var second = first;
```

Инициализиране на променливи

- Променливата се инициализира, когато ѝ бъде присвоена стойност
- Преди да бъде използвана променливата, тя трябва да получи стойност
- **Неинициализираните променливи са от тип *undefined***
- Променливите могат да бъдат инициализирани посредством:
  - Използването на литерали
  - Реферирането на вече съществуващи променливи

Видимост на променливи

- Видимостта на променливите в JS се ограничава само и единствено от фунцкия, когато променливата е декларирана чрез ключовата дума *var*
- С нововъдения стандарт ES2015 се въвежда и блокова видимост (*block scope*), но това е валидно само за последните версии на браузърите. Това може да бъде постигнато чрез използването на ключовата дума *let*
- Променливи могат да бъдат декларирани и без използването на ключовите думи *var/let*, но това е много лоша практика (променливата става глобално достъпна) и затова трябва да бъде избягвано

Променливи. Деклариране и видимост

Оператори и изрази

- Чрез операторите могат да бъдат изпълнени операции върху дадена информация по време на изпълнението на кода
- Операторите имат 1 или повече аргумента (операнда)
- Продукта от дадена операция обикновено бива нова стойност
- Операторите имат приоритет (както в математиката)
- Изразите представляват съвкупност от оператори и операнди, който се оценяват до единствена стойност

Видове оператори

- Операторите в JavaScript биват:
  - Унарни (*unary*) -> приемат само 1 операнд
  - Бинарни (binary) -> приемат 2 операнда
  - Тернарни (ternary) -> приемат 3 операнда
- Всички бинарни оператори (изкл. инициализиращите) са ляво-асоциативни
- Инициализиращите и тернарния оператор (*?:*) са дясно асоциативни

Категории оператори в JS

Категория	Оператори
Аритметични	+ - * / % ++ --
Логически	&& || ^ !
Двуични	& | ^ ~ << >>
Сравнение	== != < > <= >= === !==
Присвояване	= += -= *= /= %=
Други	. [] {} () ?:

Аритметични оператори

- Артиметичните оператори +,-,*,/ имат същото поведение, както в математиката
- % е оператор за деление с остатък, т.е. Резултата е остатъка от делението, докато при използването на / се получава цялото число
- Когато използваме оператора / за деление можем да получим 3 вида резултат: *число*, *Infinity* или *NaN*
- Специалните оператори ++, -- съответно инкрементират/декрементират променливата с единица

Логически оператори

- Логическите оператори приемат булеви операнди или операнди, които се евалюират до булеви

Операция	||	||	||	||	&&	&&	&&	&&	^	^	^	^
Операнд1	0	0	1	1	0	0	1	1	0	0	1	1
Операнд2	0	1	0	1	0	1	0	1	0	1	0	1
Резултат	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

Побитови оператори (bitwise)

- Побитовите оператори превръщат всичко в *0* и *1* (битове). Поведението им е като на логическите оператори, но работят на ниско ниво с битове
- Може да се използват операторите << и >>, за да се отметстват битове, респ. наляво или надясно

Операция	|	|	|	|	&	&	&	&	^	^	^	^
Операнд1	0	0	1	1	0	0	1	1	0	0	1	1
Операнд2	0	1	0	1	0	1	0	1	0	1	0	1
Резултат	**0**	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

Оператори за сравнение

- Операторите за сравнение действат както в математиката и се оценяват до булев резултат
- Има значение дали се използва оператора == или ===, респ. != или !==
  - ==/!= сравняват само по стойност, т.е. 5 е същото като „5“
  - ===/!== сравняват освен по стойност и по тип, т.е. 5 не е същото като „5“
- Когато очакваме *NaN* стандартните оператори за сравнение не работят и трябва да се използва предифинираната функция *isNaN(value)*

Други оператори

- `.` се използва за достъпване характеристиките/методите на даден обект
- `[]` се използват при работа с масиви или стрингове
- `{}` се използват при работата с обекти
- `()` се използват за определяне на приоритета на операциите
- `?:` се използва като условна конструкция
- Ключовата дума *new* се използва за създаването на нови обекти
- Операторът *this* се използва като референция към текущия обект

Приоритет на операторите

- Както в математиката и в други езици, в JS също има приоритет на операторите. Повече информация за приоритета на конкретните оператори може да намерите в MDN
- Препоръчва се операциите да бъдат приоритизирани чрез използването на скоби ( )
  - Сигурни сме кое кога ще се изпълни
  - Прави кода по-лесно четим и разбираем
  - Не се налага да помним/проверяваме кое след кое ще се изпълни

IT-Academy.BG

IT-Academy.BG

Домашна работа

1. Напишете JS код, който да проверява дали дадено число е четно или нечетно
2. Напишете JS код, който да проверява дали дадено число е просто.
3. Напишете JS код, който да намира лицето на окръжност по заден радиус
4. Напишете JS код, който по зададени 3 страни да може да прецени дали с тези размери може да бъде построен триъгълник