FRONTEND MĀJASLAPAS IZSTRĀDE

LATVIJAS UNIVERSITĀTE
BIZNESA, VADĪBAS UN EKONOMIKAS FAKULTĀTE

VUMC
VADĪBAS UN UZŅĒMĒJDARBĪBAS MĀCĪBU CENTRS

NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020

EIROPAS SAVIENĪBA
Eiropas Sociālais fonds

IEGULDĪJUMS TAVĀ NĀKOTNĒ

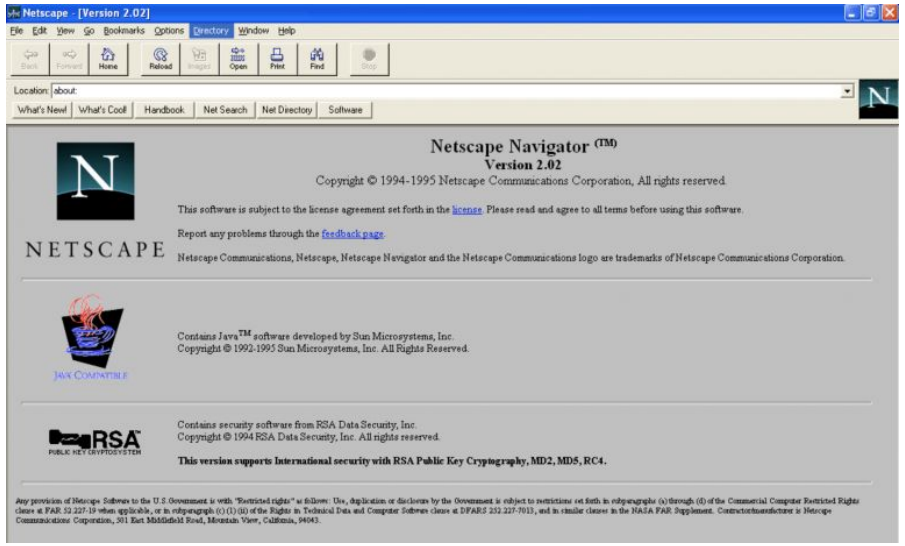
ESF projekts Nr. 8.4.1.0/16/I/001 ''Nodarbināto personu profesionālās kompetences pilnveide"

JavaScript

JavaScript aizsākums

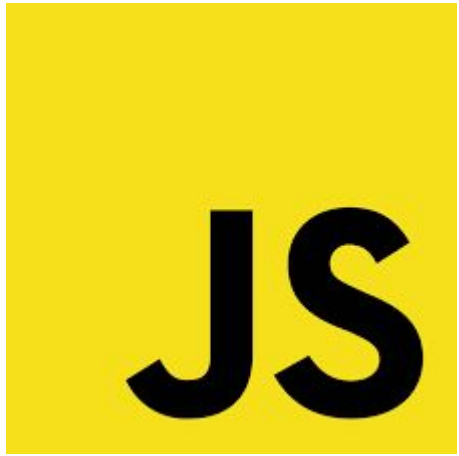
1995. gadā Netscape programmētājs Berndan Eich radīja JavaScript 10 dienu laikā.
Tās pielietojums sākotnēji bija paredzēts ļoti ierobežots un to nedēvēja par programmēšanas valodu - bet gan skriptu valodu. Netscape vēlējās padarīt savu pārlūkprogrammu pievilcīgāku lietotājam un ļaujot ieviest nelielu interaktivitāti ar mājaslapu. Tolaik Java bija ļoti populāra valoda un arī Netscape to izmantoja. Sākotēji JavaSript nosaukums bija Mocha un LiveScript, bet mārketinga nolūkos, veidot asociāciju ar Java - to pārdēvēja par JavaScript.
Citas pārlūkprogrammas to arī sāka pielietot un mūsdienās tā tiek uzskatīta par vispopulārāko programmēšanas valodu - 2021. gada aptaujā 64.96% izstrādātāju norādīja, ka profesionāli izmanto JS.

[Vissplašāk pielietotās programmēšanas valodas.]



Kas ir JavaScript?

JavaScript sava nosaukuma dēļ liek noprast par saistību ar programmēšanas valodu Java, tomēr šīs abas valodas ir ļoti atšķirīgas un nav tieši radniecīgas.
JavaScript ir uz prototipu ķēdes balstītiem objektiem orientēta skriptu jeb programmēšanas valoda. Tā neatbilst strikti vienai programmēšanas paradigmai un var pielietot vairākas (piem., objektorientētu vai funkcionālu programmēšanu).
Izmantota galvenokārt pārlūkprogrammās mājaslapu interaktivitātei. Tomēr ļoti plaši to var un pielieto kā galveno servera valodu.

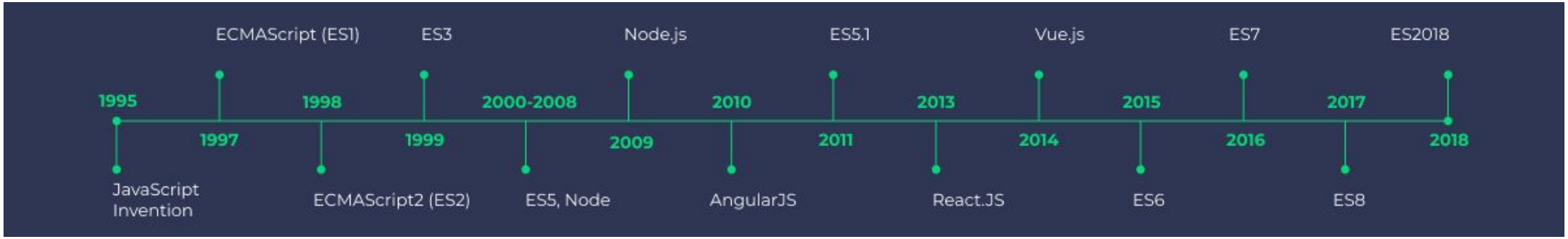


JavaScript
- multi-paradigmu programmēšanas valoda
- izpildās pārlūkprogrammās un serveros
- nav strikti definēti mainīgo tipi

Java
- objektorientēta programmēšanas valoda
- izpildās serverī
- strikti definēti mainīgo tipi

ECMAScript

ECMAScript (ES) ir JavaScript dokumentācija un standarti.
Bieži ar ECMAScript saprot - JavaScript - un lieto kā sinonīmus.
Tomēr ar ECMAScript saprotam JavaScript versiju un pieejamos rīkus, kas tiek paildināti katrā versijā.
Kopš ECMAScript 6 jeb ES6 nosaukumi ESMAScript versijām mainīja pierakstu un tagad tos apzīmē ar gadiem - aktuālākā ECMAScript versija ir ES2021.
Ar ES5 (2009. gads) un ES6 (2015. gads) JavaScript ļoti paplašināja pieejamo rīku un iespēju klāstu. ES6 tiek uzskatīts par standartu un rīku minimumu ar ko veidot modernu mājaslapu.



JS serveros - Node.Js

JavaScript sākotnējais un plašākais pielietojums ir ar pārlūkprogrammā, darboties ar mājaslapas interfeisu. Tomēr mūsdienās tas tiek ļoti plaši un veiksmīgi pielietots arī serveros ar Node.Js - atvērta pirmkoda serveru reāllaika vide, kas ļauj izpildīt JavaScript kodu servera pusē.
Netflix, Ebay, PayPal, Twitter un Uber ir tikai daži no uzņēmumiem, kuru serveri darbojās ar Node.JS vai pāriet uz to.
Īsu apskatu par izvēli Node.Js serveros šajos uzņēmumos var aplūkot šeit.

JS kosmosā

2020. gadā uzņēmums Space X nolēma savu kosmosa kuģu Dragon skārienjūtīgo ekrānu interfeisus veidot ar JavaScript.
Tā kā tie ir pirmie skārienjūtīgie ekrāni kosmoskuģī, bija nepieciešams jauns risinājums interfeisu izstrādē. Pilnu izklāstu par tā pielietojumu un priekšrocībām Space X Dragon variet izlasīt šeit.

Mainīgie un vērtības

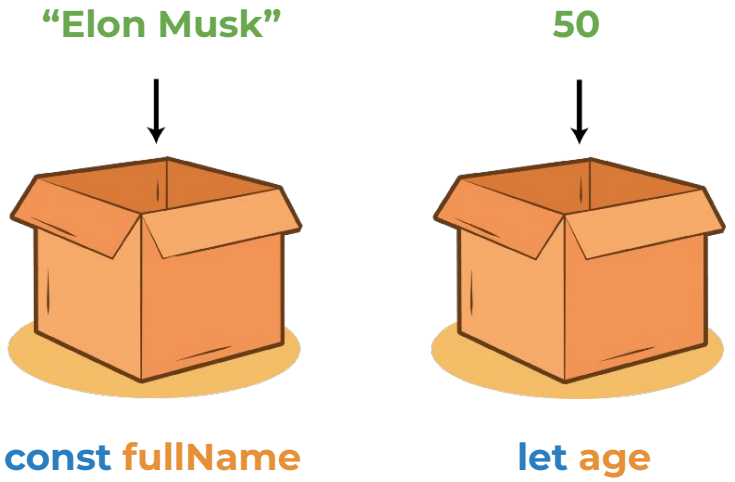
Mainīgie (variables)

Mainīgais (variable) ir vienība, kas var saturēt kādu vērtību.
JavaScript izmanto galvenokārts divus mainīgo tipus:
const - vērtība ir nemainīga konstane (ES6)
let - vērtība var tik mainīta (ES6)

Mainīgajam norādot tipu un nosaukumu, tas tiek inicializēts:
const fullName;
let age;

Mainīgajam var piešķirt vērtību:
const fullName = **“Elon Musk”**;
let age = **50**;

Daudzos interneta resursos redzēsiet mainīgā tipu **var** - tas ir novecojis un kopš ES6 ieteicams izmantot **let** vai **const**.



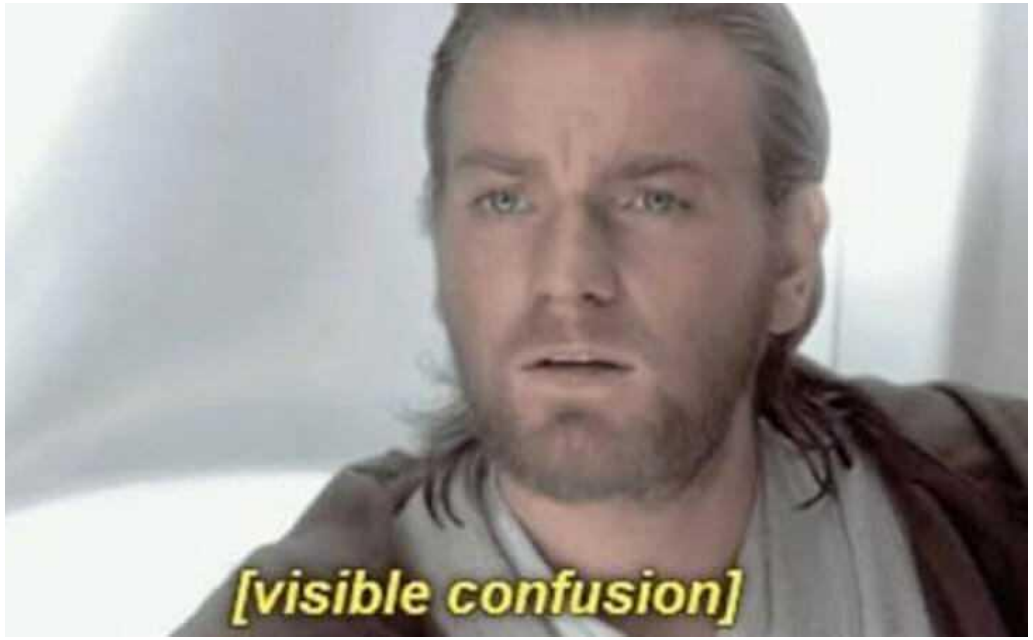
Primitīvie vērtību datu tipi

JavaScript primitīvs datu tips ir mainīgā **vērtība**, kura nav objekts un kuram ***nepiemīt metodes.**
JavaScript ir 7 primitīvās datu vērtības:

- **string**
- **number**
- **bigint**
- **boolean**
- **symbol**
- **null**
- **undefined**

***** - ērtībai darbā ar primitīvām vērtībām, JS noklusēti ietver katru primitīvo vērtību sev attiecīgā objektā, kas tomēr ļauj mums izmantot šim datu tipam atbilstošas metodes (izņemot null un undefined).

Jebkura cita vērtība, kas nav viena no šīem 7 primitīvajiem datu tipiem ir **objekts**.



string

String datu tips satur jebkādu simbolu un ciparu virkni.
Izmantojamā sintakse:

```
const doubleQuotes = ""
const singleQuotes = ''
const literals = `Some text ${someVariable}`
```

Vairāk piemēri ar string datu tipu apskatāmi W3Schools.

number

Number datu tips satur skaitlisku vērtību
no **-9007199254740991** līdz **9007199254740991**
Tas var saturēt arī abstraktas vērtības **Infinity, -Infinity, NaN**.
Kā arī vērtības **$-(2^{53} - 1)$** līdz **$(2^{53} - 1)$**.

Iespējams izmantot arī decimālvērtības - tomēr JS slavens ar savu neprecīzo aprēķinu darbaojoties ar decimālvērtībām.
Decimālvērtības iespējams noapaļot ar **toFixed()** metodi.

NaN (Not a Number) ir skaitlim neatbilstoša vērtība un par to vai vērtībā tāda ir varam pārliecināties, izmantojot metodi **isNaN()**.

```
const pi = 3.14
let year = 2022
let notReallyNumber = 0 * Infinity
```

Vairāk piemēri ar number datu tipu apskatāmi W3Schools.

bigint

BigInt var saturēt vērtības, kas ir pārāk lielas number datu tipam - veselus skaitļus lielākus par **$(2^{53} - 1)$**. Aiz skaitļa jānorāda burts **n**, lai pārlūkprogramma pareizi apstrādātu šo skaitli.
Reti izmantots datu tips, tomēr projektos ar milzīgu datu apjomu vai zinātniskiem aprēķiniem tas var būt noderīgs.

```
const incredibleNumber = 45730958349508345093485904385309583905834058304583 4n
```

boolean

Boolean var būt tikai viena no divām vērtībā:

ture vai **false**

Bieži izmantojam to kā “slēdzī” - ieslēgts vai izslēgts. Vai veicot pārbaudes, piem., divi mainīgi ir vienādi - jā vai nē.

```
const javaScriptIsAwesome = true
const aboveStatementIsNotTrue = false
```

symbol

Symbol ir datu tips, lai veidotu unikālas vērtības. Galvenokārt izmantots JS objekta slēpto īpašību nosaukumos.
Jebkuru vērtību iespējams pārvērst symbol. Šai vērtībai tiks piešķirts unikāls identifikātors ar datu tipu symbol.

```
Symbol('xyz') === Symbol('xyz') // false
```

null

Vērtība ar kuru mērķtiecīgi norādām - nav vērtība.
Bieži vien ir situācijas, kur vēlamies apzināti norādīt, ka mainīgai nesatur nekādu vērtību - tad izmantojam null.

```
const regretsInLife = null
```

undefined

Undefined ir datu tips, kas piemīt mainīgajiem, ja tiem nav piešķirta vērtība.
Arī funkcijas, kuras neatgriež vērtību - tā vietā atgriež undefined.
Bieži vien arī sastapsimiet undefined, ja kļūdaini norādīsim neeksistējoša mainīgā nosaukumu.

```
let assignMeAValue
// undefined
```

Datu tipu pārvēršana

Tā kā JavaScript nav strikti definēti mainīgo tipi tās var pārvērst no viena datu tipa citā tiešā veidā (explicit coercion):

```
String(42)
Number("42")
Boolean(42)
```

Vai netiešā veidā (implicit coercion), izmantojot dažādus operātorus, piem.,:

```
const notWhatYouExpect = 4 + 5 + 6 + ”7” // “157”
```

Citi operātori ar ko var var veikt *implicit coercion*:

&& un **||**
/ un ***** un **-** un **+**
< un **<=** un **>** un **>=**

Vairāk *implicit corecion* piemērus var apskatīt šeit.

Salīdzinājuma operātori

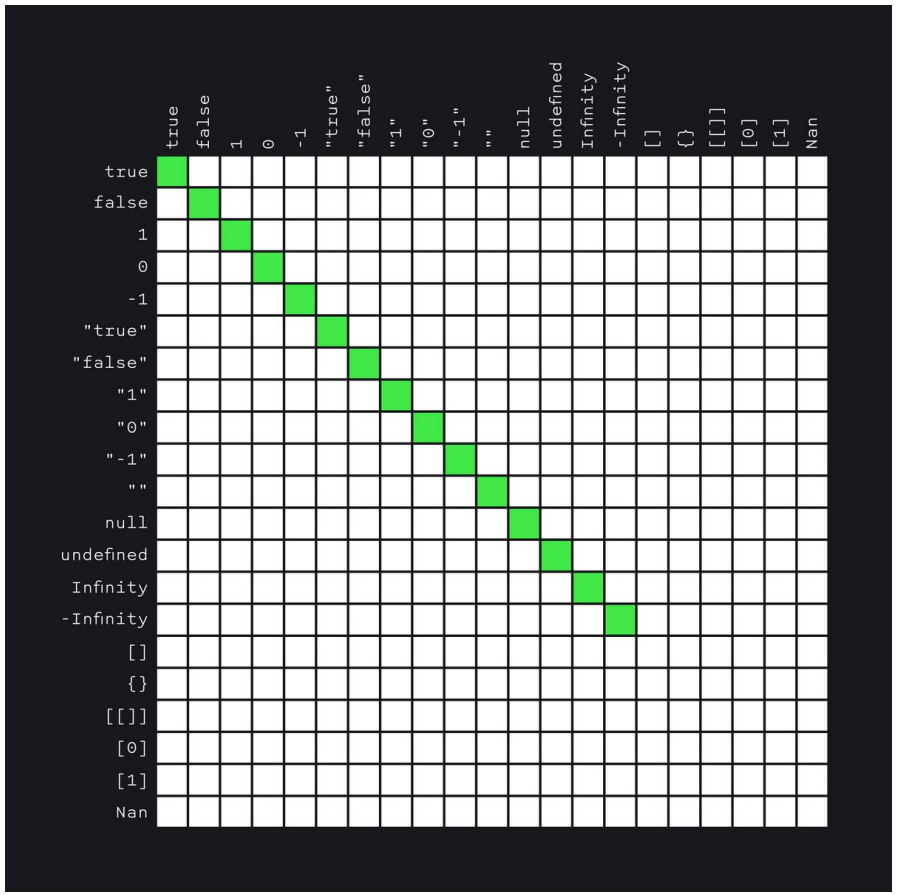
Lai salīdzinātu divu vērtību identiskumu pēc to **vertības un datu tipa**, izmantojam operātoru **===**.

"JavaScript" === "Java" // false
"0" === 0 // false
"Im starting to get it!" === "Im starting to get it!" // false

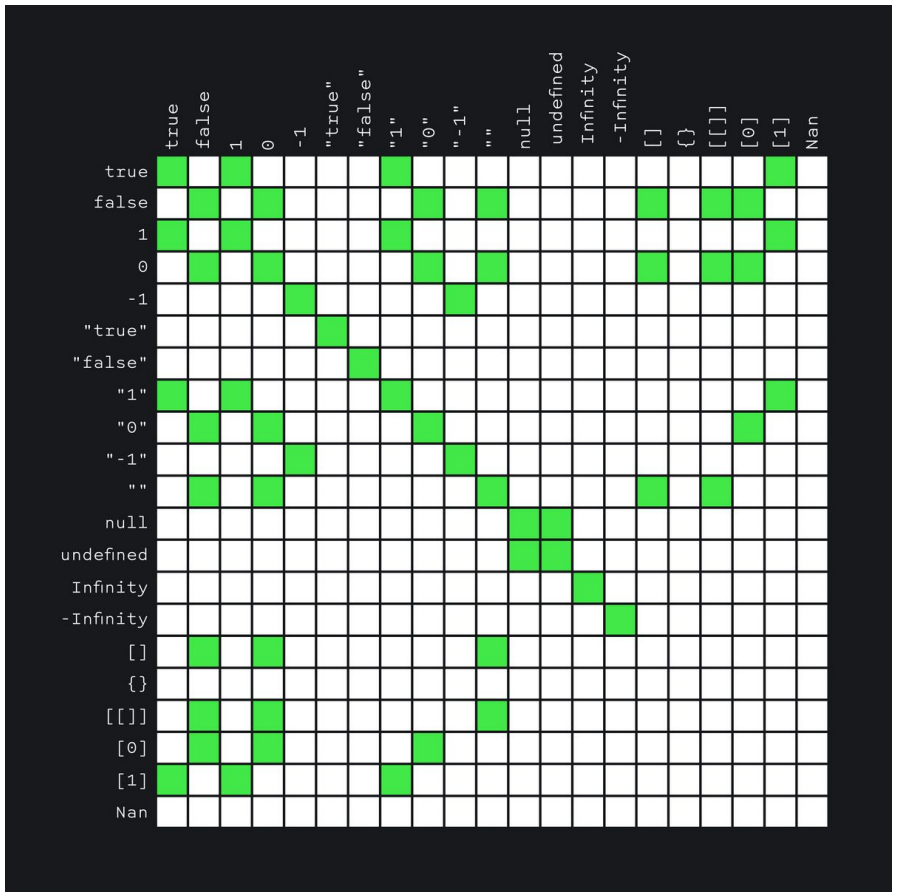
Ir piejams arī **==** salīdzinājuma operātors - tas salīdzinās tikai vērtību, bet nerūpēs datu tips. Vairums situāciju šo operātoru nav ieteicams izmanto.

"0" == 0 // true

===


true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[]
{}
[[]]
[0]
[1]
Nan

==


true
false
1
0
-1
"true"
"false"
"1"
"0"
"-1"
""
null
undefined
Infinity
-Infinity
[]
{}
[[]]
[0]
[1]
Nan

Truthy un Falsy vērtības

Visas vērtības JS ir *truthy* jeb atgriež vērtību **true**, kad tās izmanto **boolean** salīdzinājuma kontekstā, izņemot:

false
0, **-0**, **0n**
“ ” un ‘ ’
null
undefined
NaN

```
if (null) {
   // never gonna happen
} else {
  // will happen instead
}
```

Objekti (objects)

Objekts ir nosauktu vērtību apokopojums iekš viena mainīgā.
Tas var saturēt vērtības, metodes un sarežģītas struktūras.

```
const   emptyObject = { }
```

```
const   car = {
        brand: “Volvo”,
        model: “XC40”,
        doors: 4,
        airConditioner: true,
        lighting: [“LED Headlights”, “Rain Sensing Driving Lights”]
}
```

Piekļūt objekta īpašību vērtībām var izmantojot īpašību nosaukumu ķēdi:

```
automobile.model === “XC40” // true
```

Masīvs (Array)

Masīvs (array) ir kolekcija ar dažādām vērtībām.
Masīvs var sturēt tikai viena tipa vai dažādu tipu vērtības jeb emelentus.

```
const emptyArray = [ ]
const shoppingList = [“eggs”, “milk”, “bread”, “lettuce”]
const randomList = [“stuff”, 0, true, null]
```

Katram masīva elementam tiek piešķirta skaitliska vērtība sākot ar 0 pirmajam elementam. Šo skaitlisko vērtību sauc apr index.

```
shoppingList[0] === “eggs” // true
```

Darbojoties ar masīviem varam tajos meklēt, pievienot, mainīt un izņemt vērtības no tiem. Dažādas metodes ko var pielietot darba ar masīviem apskatīt šeit.

Nosacījuma izteiksmes (Conditional statements)

Bieži rakstot kodu ir nepieciešamība izpildīt atšķirīgs darbības atkarībā no kādas vērtības. Šim nolūkam izmantojam *contitional statements*:

if - norādīt koda bloku, ko veikt, ja nosacījums izpildās.
else if - cits nosacījums ar kuru norāda koda bloku, ko veikt, ja **if** nosacījums nav izpildījies.
else - norāda koda bloku, ko izpildīt, ja **if** vai **else if** nosacījum nav izpildījies.

```
if (true) {
   // do something
} else {
  // do something else
}
```

switch norāda vairākus iespējamos koda blokus ko izpildīt pie dāžādiem secīgi pārbaudītiem nosacījumiem. Vairāk piemēru atrodams W3Schools.

```
switch (variable) {
   case 0:
      // turn off
      break;
   case 1:
      // turn on
      break;
   default:
      // do something anyway
}
```

Loģiskie operātori (logical operators)

Javascript plaši pielietojam loģiskos operatorus, lai veidotu nosacījumus.

&& - un (AND)

|| - vai (OR)

! - ne (NOT)

```
if (sun === “shining” && wether === “warm”) { // go swimming }

if (sun !== “shining” || rain === true ) { // stay inside }
```

VADĪBAS UN UZŅĒMĒJDARBĪBAS MĀCĪBU CENTRS

NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020

EIROPAS SAVIENĪBA
Eiropas Sociālais fonds

IEGULDĪJUMS TAVĀ NĀKOTNĒ

ESF projekts Nr. 8.4.1.0/16/I/001 ''Nodarbināto personu profesionālās kompetences pilnveide"

VADĪBAS UN UZŅĒMĒJDARBĪBAS MĀCĪBU CENTRS

Programmas nosaukums