



→ p.1

aktivita na hodině (20)

// fakt easy, to chceš

menší úkoly (30)

// těžší, ale jde o základy

závěrečný testík (25 + 25)

// pokud sem chodíš, tak to dáš

100 \approx 4
bodů kredity

SOS!

Nikita Mironov
fit.bi.pjs@gmail.com

Vojtěch Jirkovský
jirkovoj@fit.cvut.cz

HELP!

Čím začít

<https://javascript.info/> **TOP!**

základy es5+

<https://johnresig.com/apps/learn/> **TOP!**

pokročilejší látka vysvětlující obtížná témata es5

Pokročilejší látka

<http://exploringjs.com/es6/>

co je nového v es6

<https://ponyfoo.com/>

víc lidským jazykem

...ještě dobré zdroje:

<https://github.com/getify/You-Dont-Know-JS>

<https://addyosmani.com/resources/essentialjsdesignpatterns/book/>

<https://nodejs.org/api/>

<http://dmitrysoshnikov.com/>

source of truth

<https://tc39.github.io/ecma262/>

ale nečtěte to, je to nuda

JavaScript



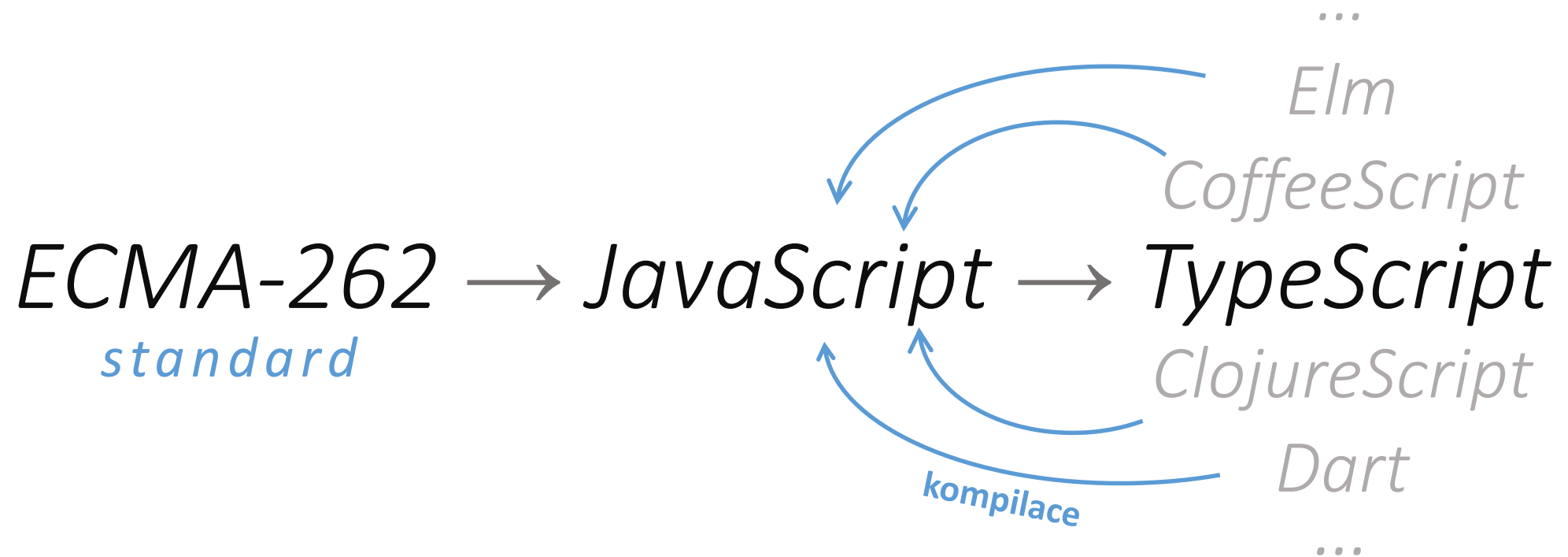
p.1 → *Úvod do JavaScriptu*

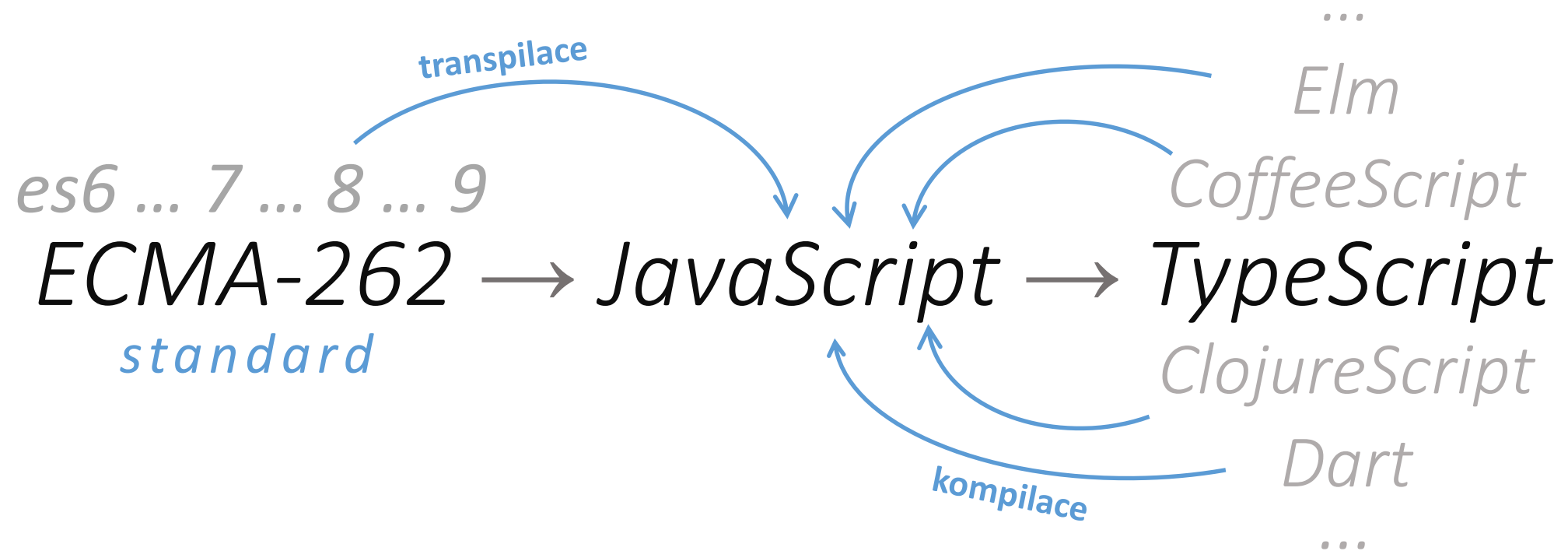
ECMA-262 → *JavaScript* → *TypeScript*
standard

JS

p.1 → Úvod do JavaScriptu

...
Elm
CoffeeScript
ECMA-262 → JavaScript → TypeScript
standard
ClojureScript
Dart
...





běžové prostředí = js engine + APIs

Firefox = spidermonkey + DOM, canvas, ...

běžové prostředí = js engine + APIs

Node.js = v8 + fs, http, zlib, ...

Vlastnosti JavaScriptu



různé operační systémy win, unix, osx, android...

multiplatformní

...chrome, firefox, ie, opera různé prohlížeče


```
// Node.js environment
const fs = require('fs') // "fs" je specificky API pro Node.js
const data = fs.readFileSync('./some-data.txt')
console.log(data)

// Browser environment
const div = document.createElement('div') // "document" je specificky API pro prohlizec.
div.innerHTML = 'ahojky'
const targetContainer = document.getElementById('container')
targetContainer.appendChild(div)
```

```
console.log(  
  'hello world'  
)
```

skriptovací
žádné binárky

01 01 10 10
00 10 01 01
11 01 00 10

větší riziko chyb

dynamický

undefined is not a function!

```
function f() {  
    inner() // funkce "inner" neexistuje, ale o tom jeste interpret nevi  
}  
  
if (Math.random() > 0.5) {  
    f()  
}
```

WAT → <https://www.destroyallsoftware.com/talks/wat>

slabě typovaný

typescript, flow, jsdoc...

```
function f(x) {  
    if (typeof x === 'number') {  
        return x * 2  
    }  
  
    if (typeof x === 'string') {  
        return 'ahoj ' + x  
    }  
  
    return 'vubec nevim co chces'  
}
```

```
function double(x) {  
    return x * 2  
}
```

```
double(2) // 4
```

```
double('2') // 4
```

určuje přesný postup



imperativní


```
const arr = [2, 7, 3]
const arr2 = []

for (var i = 0; i < arr.length; i++) {
  const doubled = arr[i] * 2
  arr2.push(doubled)
}

console.log(arr2)
```

vstupní parametry → jediný výstup + žádné vedlejší účinky

funkcionální

výpočet jako vyhodnocení funkcí

```
function double(n) {  
  return n * 2  
}
```

```
const arr = [2, 7, 3]  
const arr2 = arr.map(double)
```



jedno vlákno, jeden call stack

```
console.log('one')

setTimeout(function runMeAfterFiveSeconds() {
  console.log('two')
}, 5000)

console.log('three')
```

objektově orientovaný

prototypová dědičnost

```
const obj1 = {a: 1}
const obj2 = Object.create(obj1, {b: {value: 2}})

console.log(obj2.a) // 1
console.log(obj2.b) // 2
```

Základní syntaxe



statements

expressions

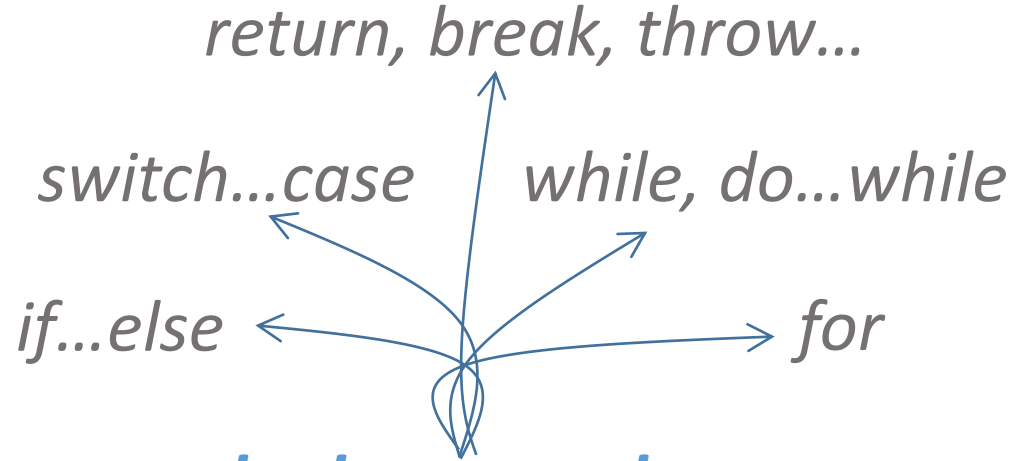
operators

statements
řídící struktury

expressions
hodnoty

operators
operace s hodnotami

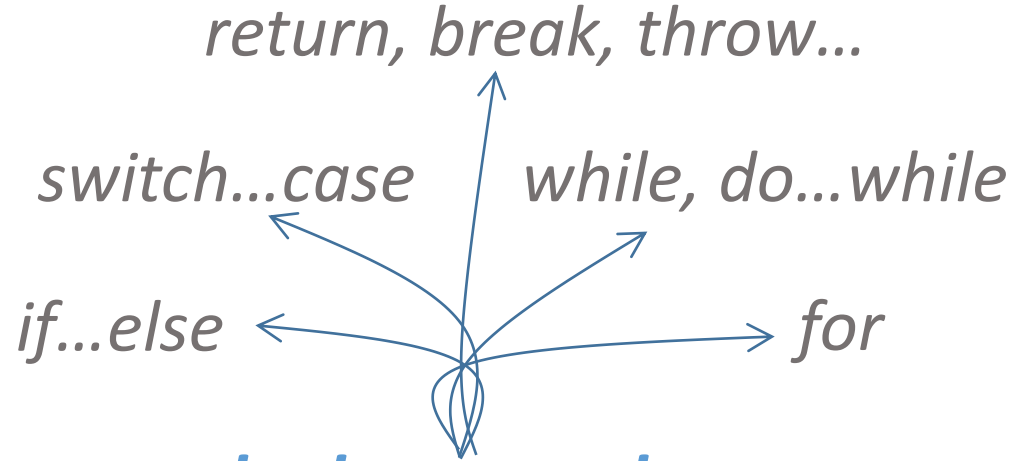




statements
řídící struktury

expressions
hodnoty

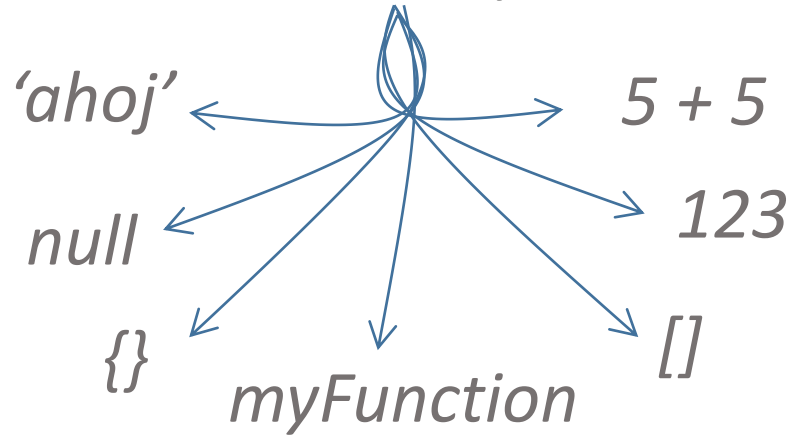
operators
operace s hodnotami

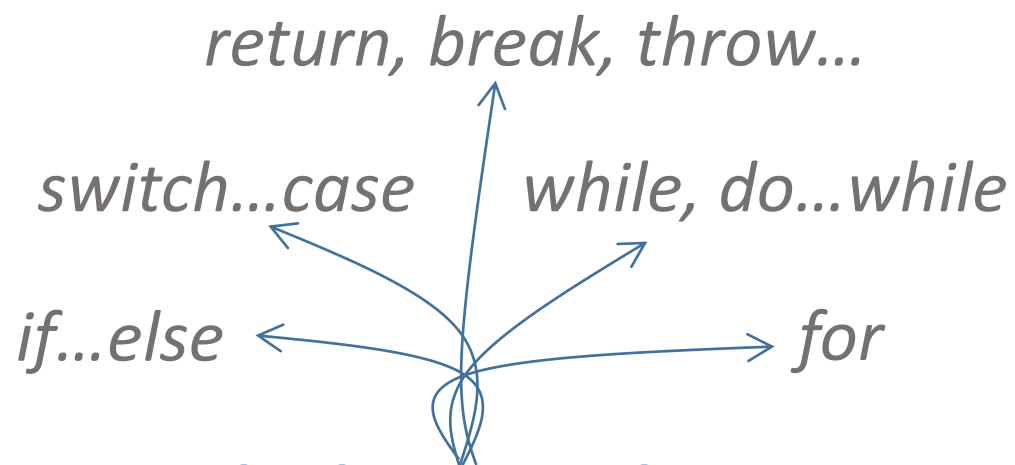


statements
řídící struktury

expressions
hodnoty

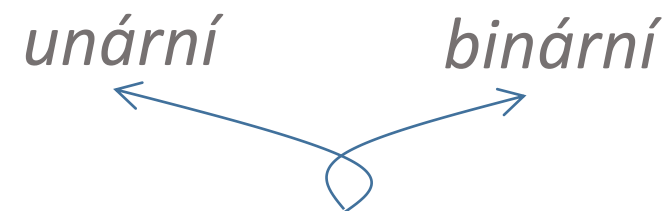
operators
operace s hodnotami



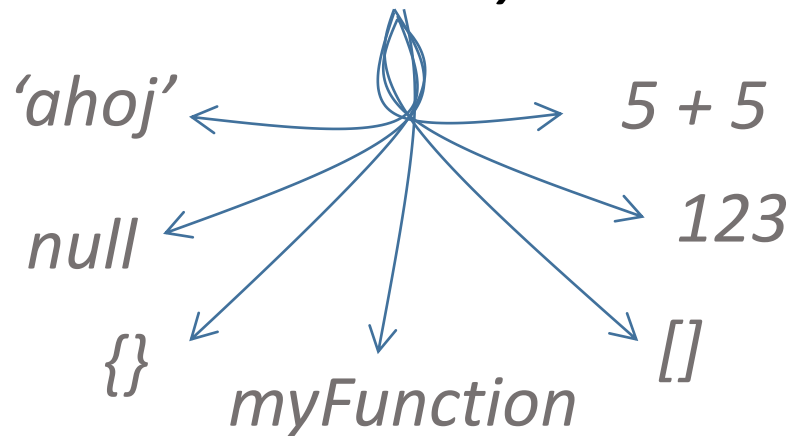


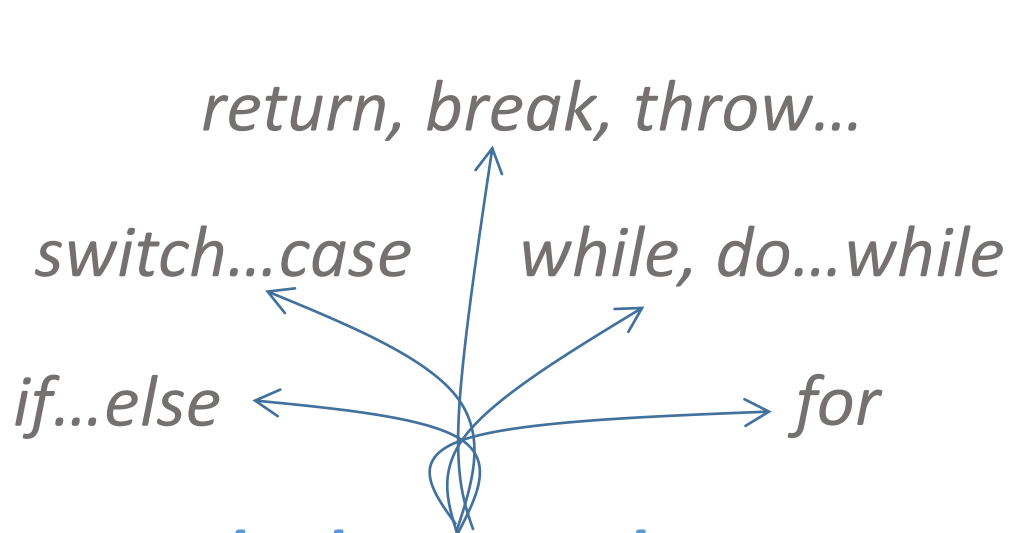
statements
řídící struktury

expressions
hodnoty

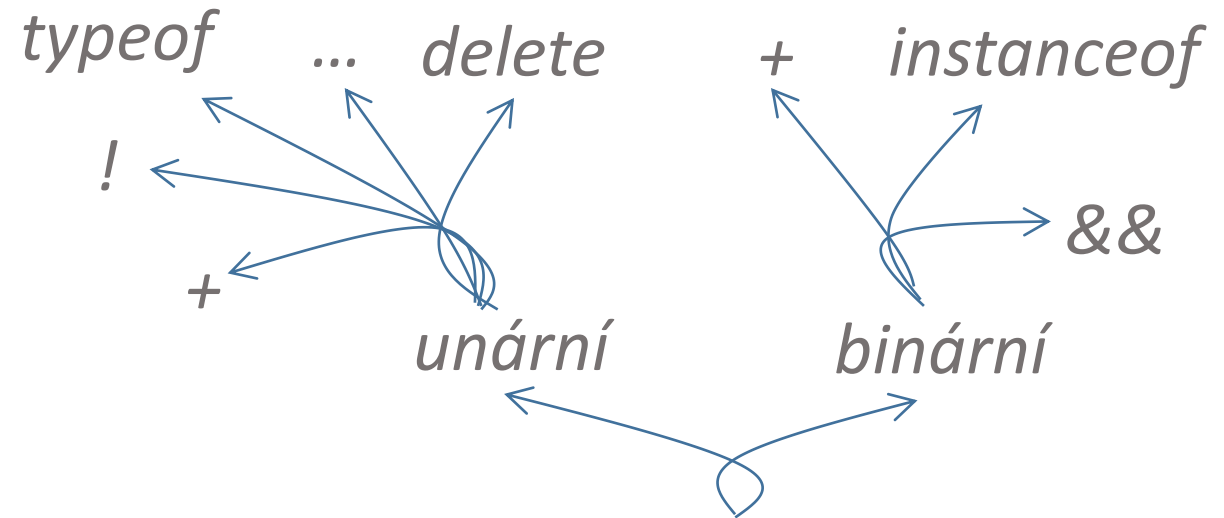


operators
operace s hodnotami



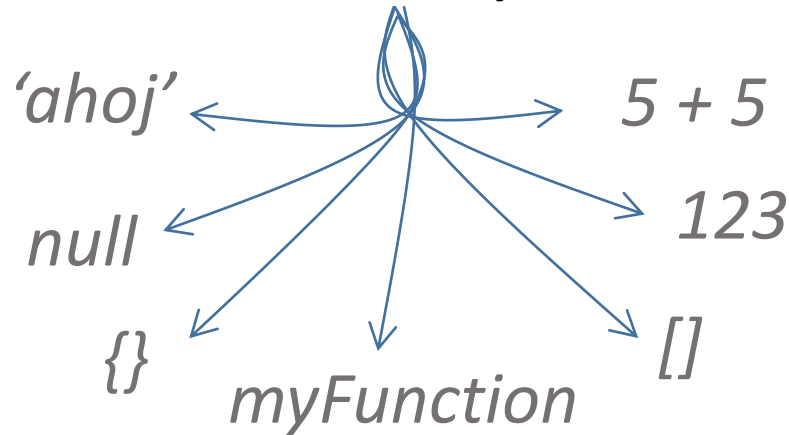


statements
řídící struktury



expressions
hodnoty

operators
operace s hodnotami



```
throw new Error('sorry jako')
```

var
let
const
function
if...else
switch
do...while
for
for...in
for...of

Statements

while
break
continue
throw
try...catch
return
class
debugger
label
block

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

var

přiřazení hodnoty do proměnné

scope: function

Utf-8 kompatibilní

case-sensitive

2semester

~~moje promenna~~

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
var a = 1  
var b = a + 1  
var c  
var d = c = b // velký špatný – zasah do globalu  
var e = 3, f = 4 // velký špatný – není to čitelný
```

```
var Petržel  
var petržel  
var $Petržel_  
var camelCase  
var snake_case  
var Zdravý Petržel  
var 1Petržel
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

let

přiřazení hodnoty do proměnné

scope: block statement

Utf-8 kompatibilní

case-sensitive

temporal dead zone

while

break

continue

throw

try...catch

return

class

debugger

label

block

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

const

přiřazení hodnoty do konstanty

scope: block statement

Utf-8 kompatibilní

case-sensitive

neblokuje změny interních hodnot objektů

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
const x // SyntaxError: Missing initializer in const declaration
```

```
const a = 1
```

```
a = 2 // TypeError: Assignment to constant variable
```

```
const b = {  
  innerProperty: 777  
}
```

```
b.innerProperty = 888 // no error
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

function

vytvoření funkce

scope: function

Utf-8 kompatibilní

case-sensitive

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
function double(n) {  
  return n * 2  
}  
  
double(2) + double(4) // 12
```


var

let

const

function

if...else

switch

do...while

for

for...in

for...of

if...else

větvení kódu

vyhodnocení truthy / falsy hodnot

imperativní obdoba ternárního operatoru

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
if (Math.random() > 0.5)
  console.log('jsem vetsi nez 0.5')

if (Math.random() > 0.5)
  console.log('jsem vetsi nez 0.5')
  console.log('a to je urcite pravda') // toto se provede v kazdem pripade

if (Math.random() > 0.5) { // <- sexy zavorky navíc
  console.log('jsem vetsi nez 0.5')
  console.log('a to je urcite pravda')
} // <- tu take

if (Math.random() > 0.5) {
  console.log('jsem vetsi nez 0.5')
  console.log('a to je urcite pravda')
} else {
  console.log('jsem mensi nez 0.5')
}
```

```
const n = Math.random()

if (n > 0.5) {
  console.log('jsem vetší')
} else {
  if (n < 0.5) {
    console.log('jsem menší')
  } else {
    console.log('jsem 0.5')
  }
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

switch

větvení kódu na základě hodnoty

dynamic cases

stejná větev pro více hodnot

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
const n = Math.floor(Math.random() * 10) // nahodne cislo od 0 do 9

switch (n) {
  case 1: {
    console.log('jsem jednicka')
    break
  }
  case 2: case 3: {
    console.log('jsem dvojka nebo trojka')
    break
  }
  case 4: {
    console.log('jsem ctyrka')
  }
  case 5: {
    console.log('jsem ctyrka nebo petka')
    break
  }
  default: {
    console.log('jsem nula nebo sestka nebo sedmicka nebo osmicka nebo devitka')
  }
}
```

```
const n = 1

switch (n) {
  case 1: {
    console.log('jsem jednicka')
    break
  }
  case '1': {
    console.log('jsem string a proto se neprovedu')
    break
  }
}
```

```
const n = 1.23
```

```
switch (n) {  
  | case 1 + 0.23: {  
    |   console.log('magie, kterou radši nepoužívát')  
    |   break  
    | }  
  |  
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

do...while

cyklus s jednou podmínkou

zaručeně alespoň jedna iterace

while

break

continue

throw

try...catch

return

class

debugger

label

block


```
do {  
  console.log('urcite provedu se aspon jednou')  
} while (false)
```

```
let i = 5  
do {  
  console.log(i)  
} while (i--)
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

for

cyklus s inicializací, podmínkou a finálním expression

for (

[inicializace];

[podmínka];

[finální expression]

) { tělo cyklu }

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
for (let i = 0; i < 10; i++) {  
  console.log(i)  
}  
  
for (;;) {  
  console.log('nekonecny cyklus')  
}
```

```
for (1;2;3) {  
    // ...  
}  
  
// ...je stejný cyklus jako...  
  
1  
while (2) {  
    // ...  
    3  
}
```

```

function checkPrime(n){
  if (typeof n !== 'number' || !Number.isFinite(n)){
    return false
  }
  n = Math.abs(n)
  if (((n === 1) || (n % 1)) || ((n > 2) && (!(~~(n % 10) % 2))) || ((n > 5) && (~~(n % 10) === 5))) {
    return false
  }
  // priklad hodne necitelnyho foru
  for (var i = 3, end = ~~Math.pow(n, 1/2), a = [4, 2, 2, 2], j = 0; i <= end; i += a[j++ % 4]) {
    if (!(n % i)) return false
  }
  return true
}

```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

for...in

cyklus iterující nad klíči objektu

```
for (  
    var key in object  
) { tělo cyklu }
```

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
const obj = {  
  a: 1,  
  b: 2  
}  
  
for (var key in obj) {  
  console.log('klic', key)  
  console.log('hodnota', obj[key])  
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

for...of

cyklus využívající interní iterátor objektu

```
for (  
    var value of object  
) { tělo cyklu }
```

while

break

continue

throw

try...catch

return

class

debugger

label

block


```
const arr = [2, 3, 4]
```

```
for (const value of arr) {  
  console.log(value)  
}
```

```
// typeof arr[Symbol.iterator] === 'function'
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

while

cyklus s podmínkou

```
while (  
    x < 10  
) { x++ }
```

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
let i = 10
while (i--) {
  console.log(i)
}
// 9
// ...
// 0

while (true) { // nebo jakakoliv truthy hodnota
  console.log('nekonecny cyklus')
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

break

zastavuje cyklus

může zastavovat vnější cykly pomocí labelů

```
while (  
    x < 10  
) {  
    if (x === 7) { break }  
    x++  
}
```

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
let i = 10
while (i--) {
  if (i === 3) {
    break
  }
  console.log(i)
}
// 9
// ...
// 4
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

continue

pokračuje na další iteraci

také podporuje labely

```
while (  
    x < 10  
) {  
    if (x === 7) { continue }  
    x++  
}
```

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
let i = 10
while (i--) {
  if (i === 3) {
    continue
  }
  console.log(i)
}
// 9
// ...
// 4
// 2
// 1
// 0
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

throw

vyhazuje vyjímku (nastala chyba)

podporuje jakýkoliv expression

zastaví běh programu pokud není uvnitř try...catch

throw new Error('stala se chyba')

while

break

continue

throw

try...catch

return

class

debugger

label

block


```
let i = 10
while (i--) {
  if (i === 3) {
    throw new Error('velikanska chyba')
  }
  console.log(i)
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

try...catch

ošetření & zachycení výjimky

try...catch...finally

neplatí pro SyntaxError

```
try {  
    throw new Error('testovací chyba')  
} catch(err) {  
    console.log('zachytili jsme chybu', err)  
}
```

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
try {  
  let i = 10  
  while (i--) {  
    if (i === 3) {  
      throw new Error('velikanska chyba')  
    }  
    console.log(i)  
  }  
} catch(err) {  
  console.log('zachytil jsem chybu', err)  
  console.log('ale vim co s ni, jedu dal')  
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

return

vrací výsledek funkce

zastaví funkci

citlivý na EOL

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
function double(n) {  
    return n * 2  
}  
  
function triple(n) {  
    return n * 3  
    return n * 2 // unreachable code  
}
```

```
function doSomething(n, m) {  
  if (n > 5) {  
    return n * m  
  } else {  
    return n + m  
  }  
}
```

```
function doSomething(n, m) {  
  if (n > 5) {  
    return n * m  
  }  
  
  return n + m  
}
```

```
function multiply(n, m) {  
    return  
        n * m  
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

class

vytváří třídu

syntaktický cukr pro funkci s prototypem

podporuje třídní dědičnost

```
class Cat extends Animal {  
  constructor(name) {  
    this.name = name  
  }  
}
```

while

break

continue

throw

try...catch

return

class

debugger

label

block


```
class Animal {
  constructor(name) {
    this.name = name
  }
  sayName() {
    console.log(this.name)
  }
}

class Cat extends Animal {
  sayName() {
    console.log('meow, moje jmeno je ' + this.name)
  }
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

debugger

zastaví běh programu v určitém místě pro debugování

= breakpoint

```
function f(a) {  
    a = a * a  
    debugger  
    return a + 100  
}  
f(5)
```

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
function doSomething(n, m) {  
  const x = n * m  
  if (x > 10) {  
    return 'vetsi nez deset'  
  }  
  return 'mensi nebo rovno deset'  
}  
  
doSomething(3, 5)
```

```
function doSomething(n, m) {  
    const x = n * m  
    debugger // zastavime beh a podivame se co je aktualne v "x"  
    if (x > 10) {  
        return 'vetši nez deset'  
    }  
    return 'mensi nebo rovno deset'  
}  
  
doSomething(3, 5)
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

label

odkaz na cyklus nebo blok kódu

používá se společně s break/continue

while

break

continue

throw

try...catch

return

class

debugger

label

block

```
let i = 10
outerCycle: while (i--) {
  let j = 10
  while (j--) {
    if (i * j === 49) {
      break outerCycle
    }
  }
}
```

```
function doSomething(n) {  
  squareRootBlock: {  
    if (n < 0) {  
      break squareRootBlock  
    }  
    n = Math.sqrt(n)  
  }  
  
  return n + 5  
}
```

var

let

const

function

if...else

switch

do...while

for

for...in

for...of

block

blók kodu (zužující scope pro let a const)

```
function f(x) {  
    let a = 1  
    let b = 2  
    {  
        let b = 3  
    }  
    return a + b  
}
```

while

break

continue

throw

try...catch

return

class

debugger

label

block

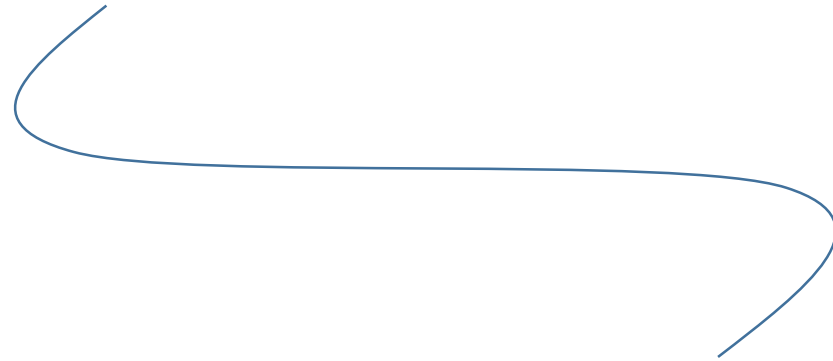

```
function f() {  
  let a = 1  
  let b = 2  
  {  
    let b = 3  
  }  
  return a + b  
}
```

Operators



p.1 → *Základní syntaxe* → *Operators*

left-to-right, right-to-left



vlastnosti → směr, priorita

```
// right-to-left  
var a = b = c = 3  
  
// left-to-right  
var b = 1 + 2 + 3  
  
// nejdriv &&, potom ||  
var c = true && false || true && true
```

druhy → unární, binární, ternární

```
// unarni operator
+'1' // unarni plus – prevod na cislo
delete obj.key // obj.key === undefined

// binarni operator
1 + 2 // 3
window instanceof Object // true

function isBiggerThanFive(n) {
    // ternarni operator
    return n > 5 ? true : false // nebo jednoduse return n > 5
}
```

semantika



aritmetické, logické, relační, bitové

= — *přiřazení*

() — prioritizace nebo volání funkce

```
function double(n) {  
  return n * 2  
}  
  
console.log(double) // [Function: double]  
// volani funkce  
console.log(double(5)) // 10  
  
// prioritizace  
console.log((5 + 4) * 3) // 27  
  
// expression grouping  
console.log(1, 2, 3, 4) // 1, 2, 3, 4  
console.log((1, 2, 3, 4)) // 4
```

*+ — sčítání (1 + 2)
konkatenace ('a' + 'b')
převod na číslo (+'2')*

```
console.log(+ '123') // 123  
console.log(+ 'a') // NaN  
console.log(1 + 2) // 3  
console.log('a' + 'b') // 'ab'  
console.log('1' + 2 + 3 + 4) // '1234'  
console.log(1 + 2 + 3 + '4') // '64'
```

- — *odečítání*
- * — *násobení*
- / — *dělení*
- % — *modulo*

`++a` — pre-inkrementace

`a++` — post-inkrementace

`--a` — pre-dekrementace

`a--` — post-dekrementace

```
var a = 5  
var b = 3 + a++  
console.log(a, b) // 6, 8
```

```
var a = 5  
var b = 3 + ++a  
console.log(a, b) // 6, 9
```

```
var a = 5  
var b = 3 + a--  
console.log(a, b) // 4, 8
```

```
var a = 5  
var b = 3 + --a  
console.log(a, b) // 4, 7
```

`==`, `!=` — *nestriktní porovnání*

`===`, `!==` — *striktní porovnání*

`>`, `>=` — *větší, větší nebo rovno*

`<`, `<=` — *menší, menší nebo rovno*


```
if (1 == '1') {  
  console.log('porovnani bez ohledu na datove typy operandu')  
}  
  
if (1 === '1') {  
  console.log('porovnani s ohledem na datove typy operandu')  
}
```

	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																					



! — *negace*

&& — *konjunkce*

// — *disjunkce*

výsledkem je jeden z operandů!

! — negace

&& — konjunkce

// — disjunkce

```
const a = false  
const b = true  
console.log(!a) // true  
console.log(a && b) // false  
console.log(a || b) // true
```

```
console.log(1 && 'ahoj' && {} && 0 && true) // 0  
console.log(0 || NaN || undefined || 'ahoj' || 5) // 'ahoj'
```

$a ? b : c$ — ternární

vymazání klíče z objektu

je instancí dané třídy nebo ne

delete, typeof, instanceof

zjištění typu proměnné

void, &, /, ^, ~, <<, >>, >>>, ,

Komentáře



p.1 → *Základní syntaxe* → *Komentáře*

// one line

*/**

multi-line

**/*

Úkoly → bit.ly/2EmhzFX

// end