→ p.1

aktivita na hodině (20)

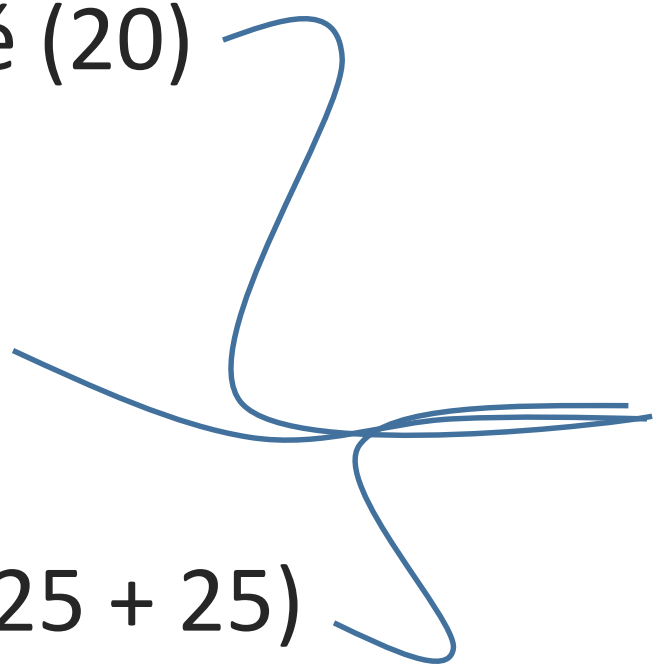
// fakt easy, to chceš

menší úkoly (30)

// těžší, ale jde o základy

závěrečný testík (25 + 25)

// pokud sem chodíš, tak to dáš



100 bodů ≈ 4 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

ECMA-262 → *JavaScript* → *TypeScript*

standard

ECMA-262 → *JavaScript* → *TypeScript*

standard

…

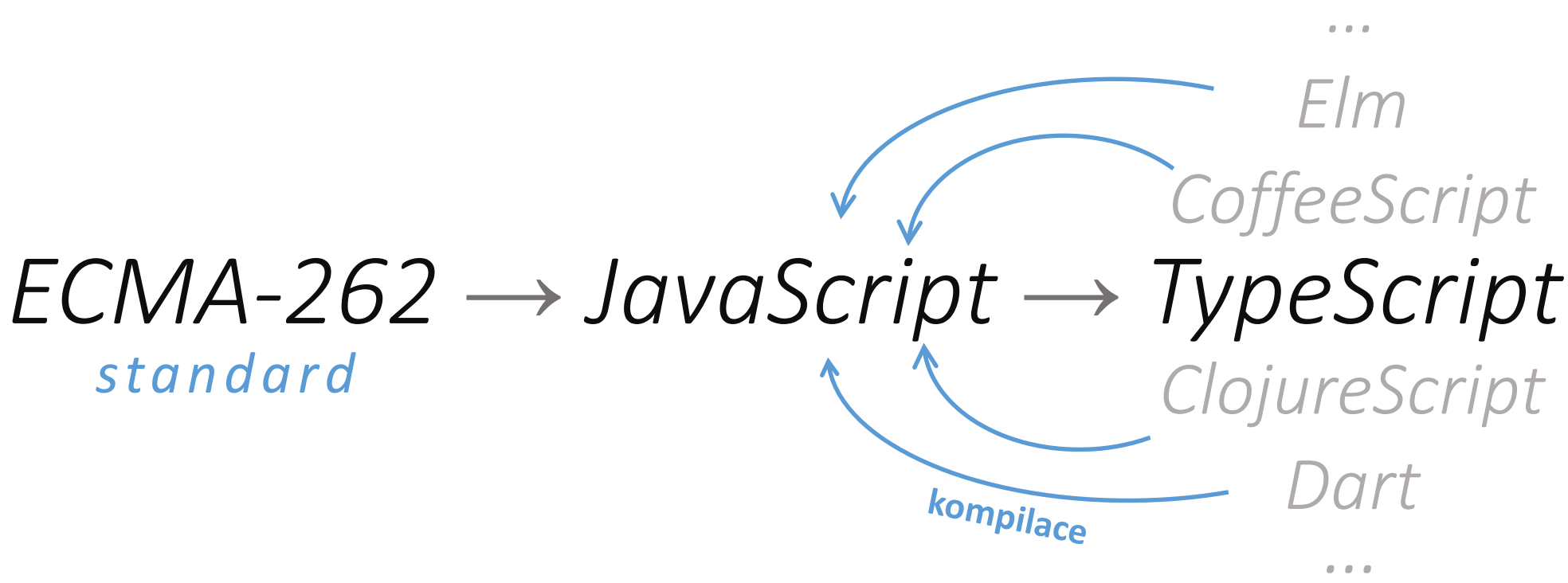
Elm

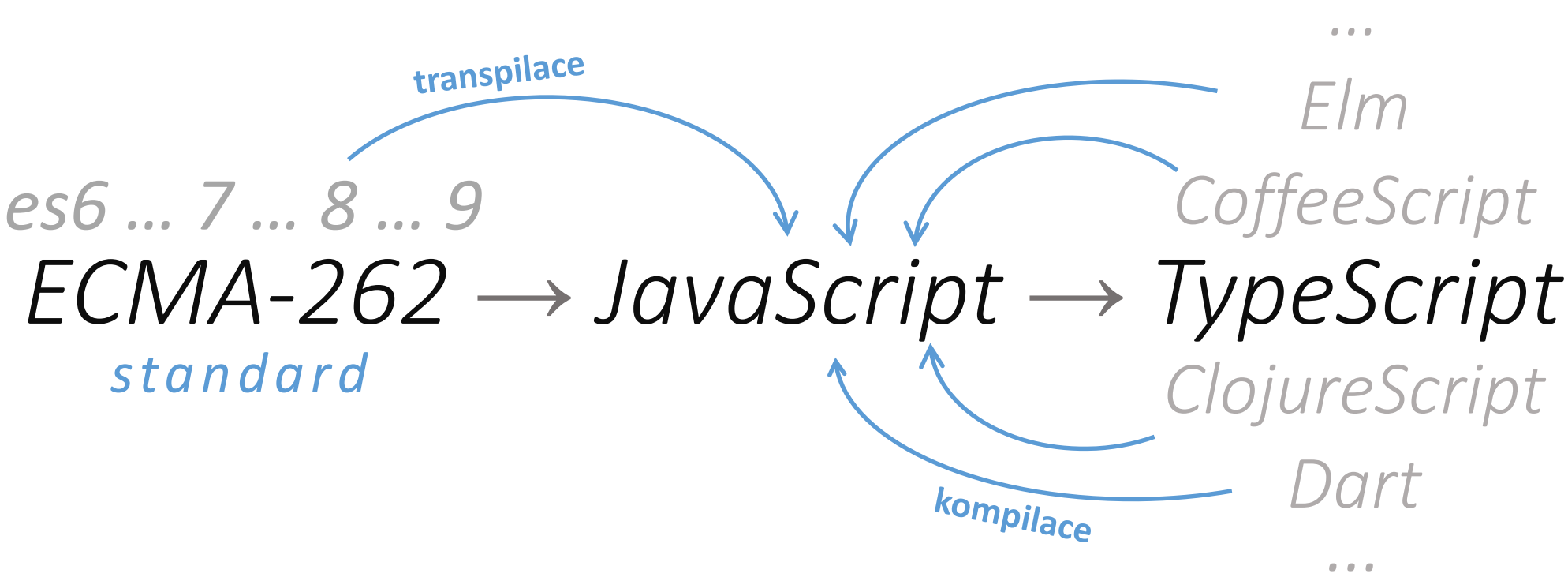
CoffeeScript

ClojureScript

Dart

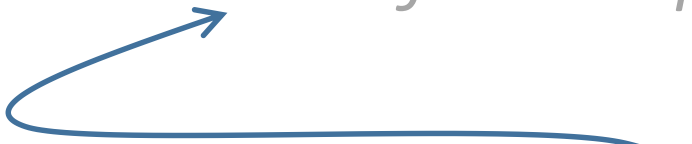
…


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


transpilace
es6 … 7 … 8 … 9
ECMA-262
standard
JavaScript
TypeScript
…
Elm
CoffeeScript
ClojureScript
Dart
…
kompilace

běhové prostředí = js engine + APIs

Firefox = spidermonkey + DOM, canvas, …



běhové prostředí = js engine + APIs

Node.js = v8 + fs, http, zlib, …

Vlastnosti JavaScriptu

různé operační systémy win, unix, osx, android…

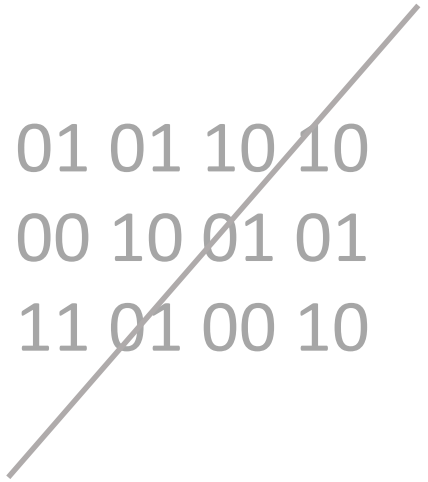
multiplatformní

…chrome, firefox, ie, opera různé prohlížeče

```
console.log(
        ‘hello world’
)
```

skriptovací

žádné binárky



větší riziko chyb

dynamický

undefined is not a function!

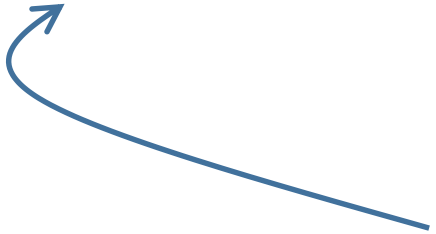
WAT → https://www.destroyallsoftware.com/talks/wat

slabě typovaný

null + 1 – [] === 1

typescript, flow, jsdoc…

určuje přesný postup



imperativní

```
for (var i = 0; i < arr.length; i++) arr2.push(arr[i] * 2)
```

výpočet jako vyhodnocení funkcí

vstupní parametry ⟶ jediný výstup + žádné vedlejší účinky

funkcionální

```
const arr2 = arr.map(value => value * 2)
```

jedno vlákno, jeden call stack

objektově orientovaný

prototypová dědičnost

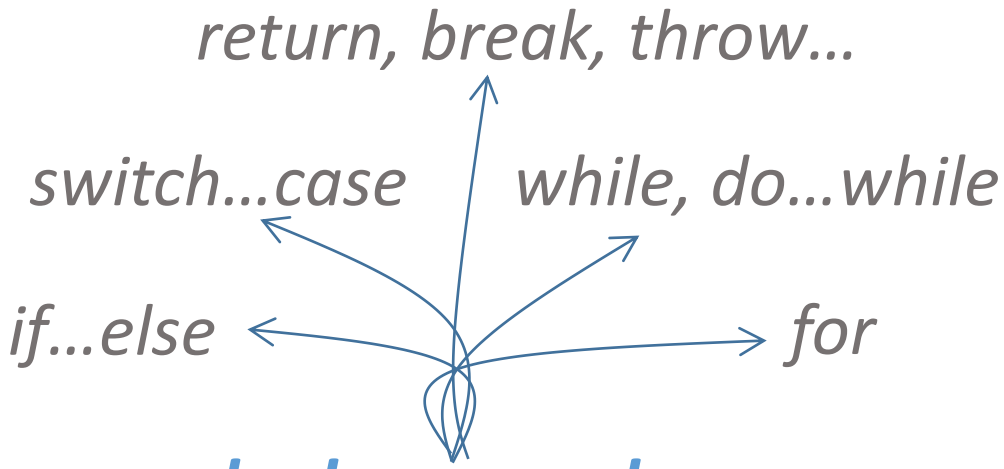
Základní syntaxe

statements *expressions* *operators*

statements
řídicí struktury

expressions
hodnoty

operators
operace s hodnotami



statements

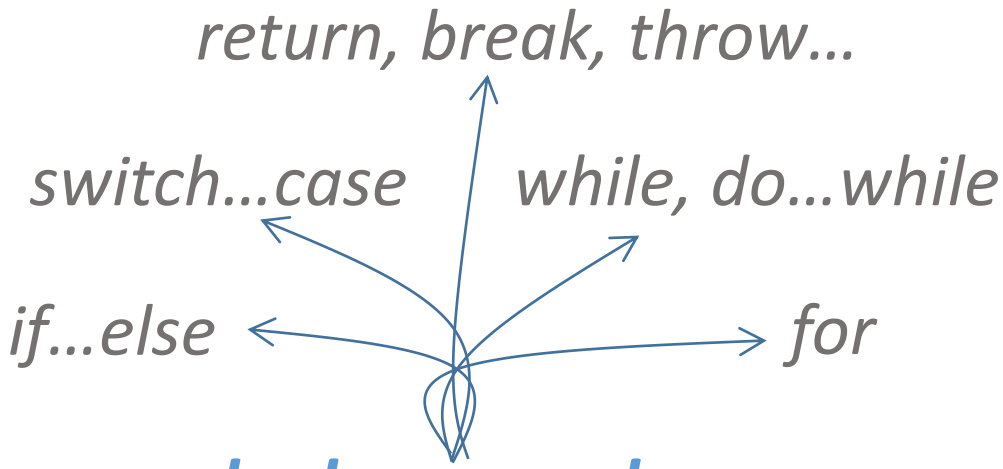
řídicí struktury

expressions

hodnoty

operators

operace s hodnotami

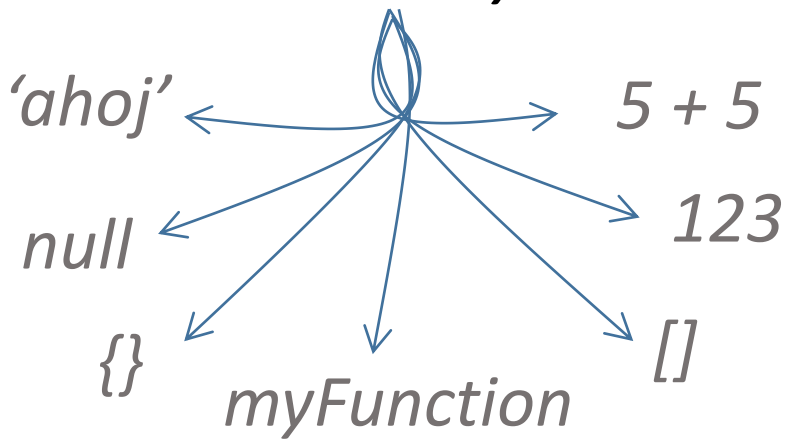


statements

řídicí struktury

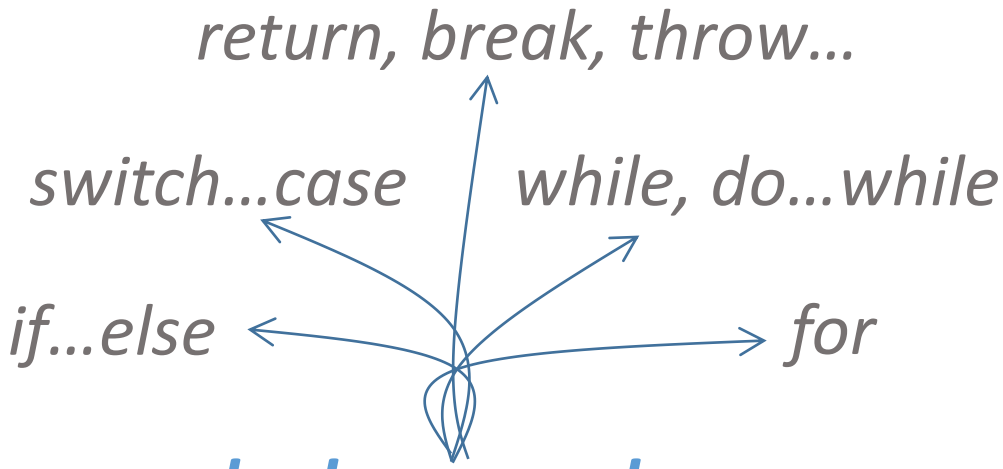
expressions

hodnoty



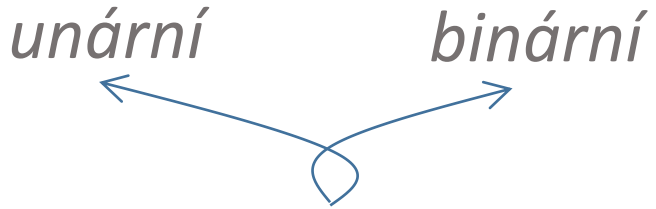
operators

operace s hodnotami



statements

řídicí struktury

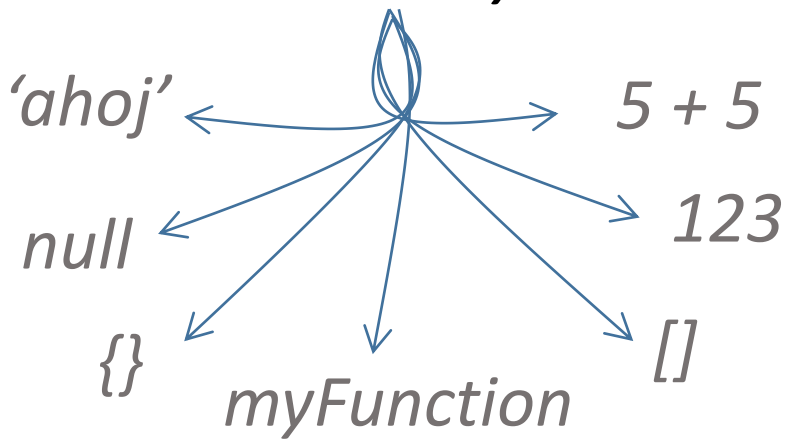


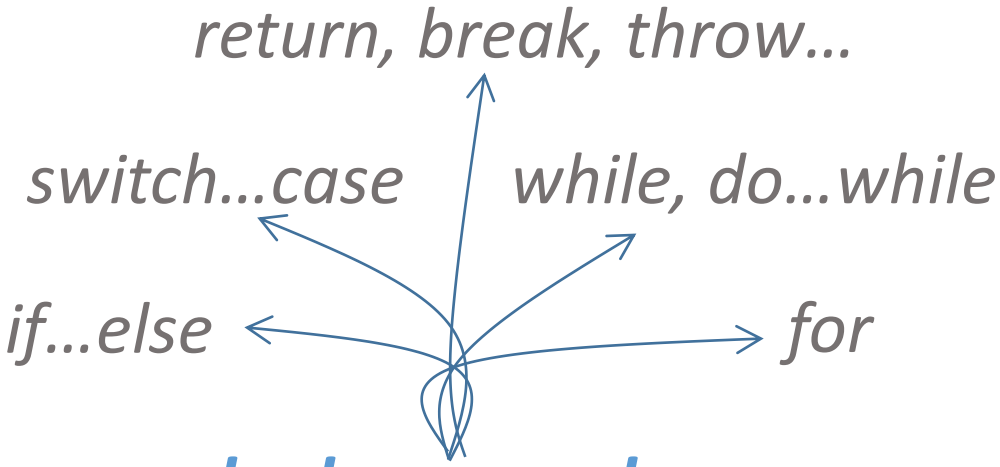
operators

operace s hodnotami

expressions

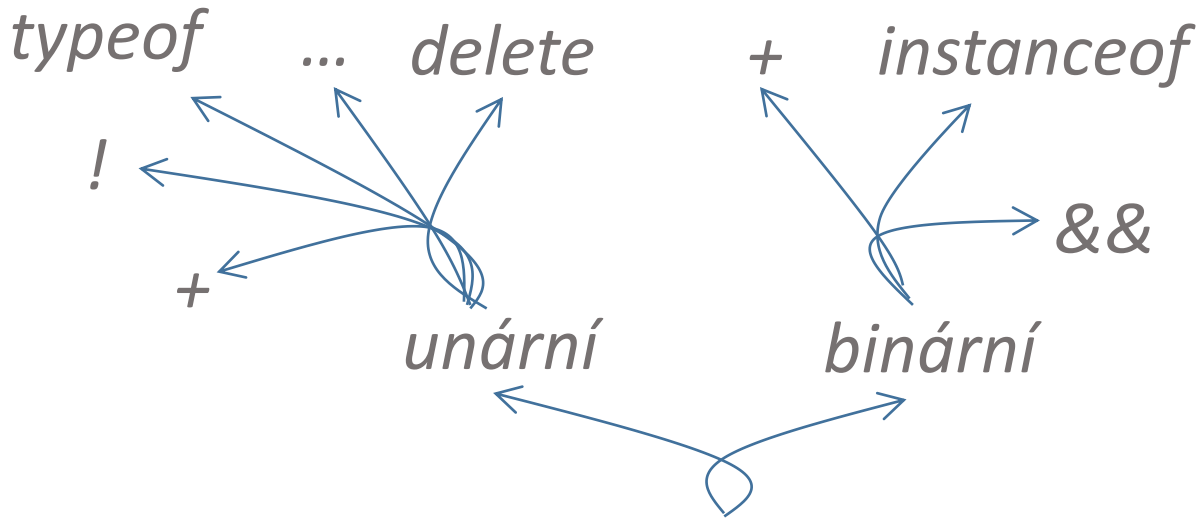
hodnoty





statements

řídicí struktury

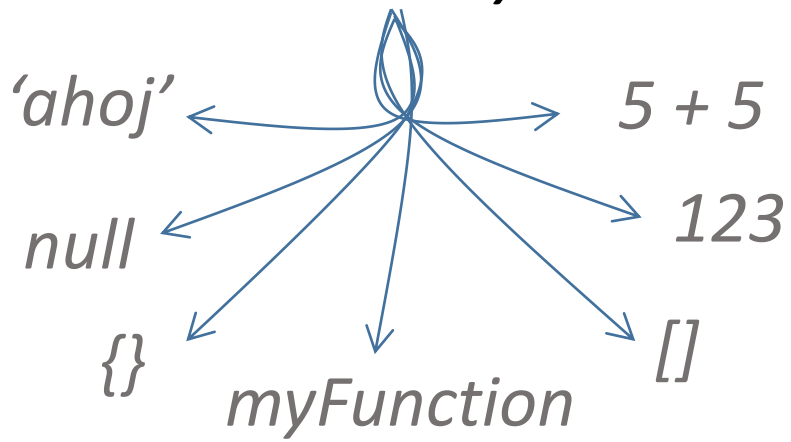


operators

operace s hodnotami

expressions

hodnoty



Statements

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

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
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

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

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

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

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

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

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

for

cyklus s inicializaci, podmínkou a finálním expression

```
for (
        [inicializace];
        [podmínka];
        [finální expression]
) { tělo cyklu }
```

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

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

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

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

break

zastavuje cyklus

může zastavovat vnější cykly pomoci labelů

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

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

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

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 neni uvnitř try...catch

throw new Error('stala se chyba')

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

try...catch

ošetření & zachycení výjimky

try...catch...finally

neplatí pro SyntaxError

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

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

return

vrací výsledek funkce

zastaví funkci

citlivý na EOL

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

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

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

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

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

block

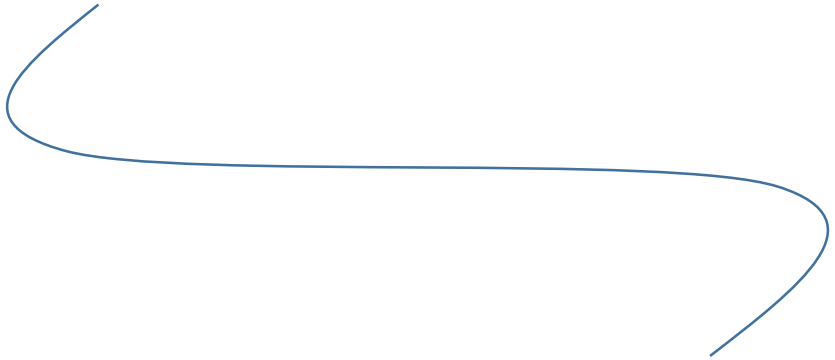
blók kodu (zužující scope pro let a const)

```
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

Operators

left-to-right, right-to-left



vlastnosti → směr, priorita

druhy → unární, binární, ternární

semantika

↓

aritmetické, logické, relační, bitové

= — *přirovnání*

() — prioritizace nebo volání funkce

+ — sčítání (1+2)

konkatenace (‘a’ + ‘b’)

převod na číslo (+’2’)

- — odečítání

** — násobení*

/ — dělení

% — modulo

++a — pre-inkrementace

a++ — post-inkrementace

--a — pre-dekrementace

a-- — post-dekrementace

==, != — nestriktní porovnání

===, !== — striktní porovnání

>, >= — větší, větší nebo rovno

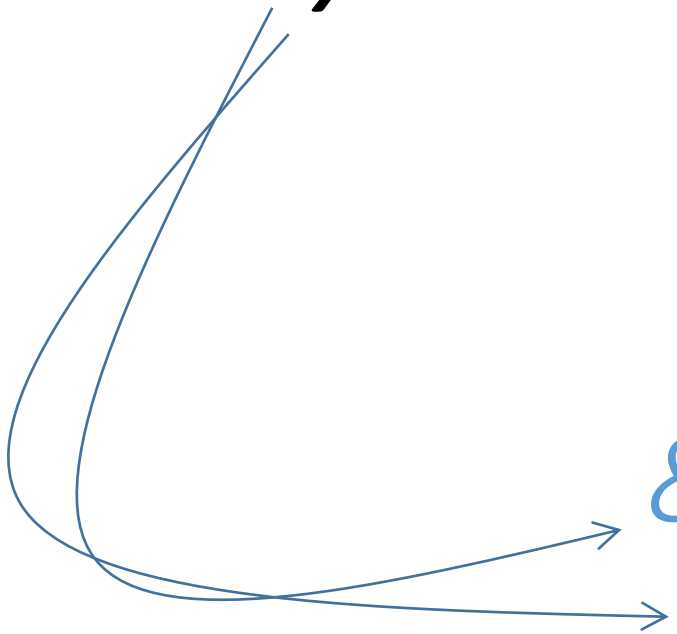
<, <= — menší, menší nebo rovno

! — *negace*

&& — *konjunkce*

|| — *disjunkce*

výsledkem je jeden z operandů!



! — *negace*

&& — *konjunkce*

|| — *disjunkce*

a ? b : c — ternární

vymazání klíče z objektu

je instancí dané třídy nebo ne

delete, *typeof*, *instanceof*

zjištění typu proměnné

void, *&*, *|*, *^*, *~*, *<<*, *>>*, *>>>*, *,*

Komentáře

```
// one line
```

```
/*
    multi-line
*/
```

(1b) *Napište aspoň 4 vlastností JavaScriptu.*

(2b) *Převeďte následující kód na kód, využívající cyklus while.*

```
for (var i = 0; i < 5; i++) console.log(i)
```

// end