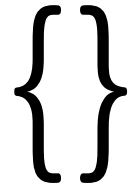
→ p.3

Object

p.3 → *Object jako datová struktura*



{*key*: *value*}

{*key*: *value*}

object ≈ *“slovník”* ≈ *“mapa”*

{*key*: *value*}

object ≈ "slovník" ≈ "mapa"

libovolný počet párů klíč — hodnota

```
/* objekt, obsahujici dve vlastnosti - a, b */
const obj = {
    a: 1,
    b: 2
}

/* uprava vlastnosti */
obj.a = 3

/* vnorene objekty (neni to dedicnost) */
const obj2 = {
    x: 3,
    y: 2,
    z: {
        a: 1,
        b: 4
    }
}
```

vlastnost může být vlastní nebo zděděná

Own

↓

vlastnost může být vlastní nebo zděděná

...Object.getOwnPropertyNames...

...Object.getOwnPropertyDescriptors...

...hasOwnProperty

Own

↓

vlastnost může být vlastní nebo zděděná

...Object.getOwnPropertyNames...

...Object.getOwnPropertyDescriptors...

...hasOwnProperty

Own

↓

vlastnost může být vlastní nebo zděděná

↑

{a: 1, b: 2, c: 3}

...Object.getOwnPropertyNames...

...Object.getOwnPropertyDescriptors...

...hasOwnProperty

Own

↓

vlastnost může být vlastní nebo zděděná

↑

{a: 1, b: 2, c: 3}

≈

{a: 1} → {b: 2} → {c: 3}

...Object.getOwnPropertyNames...

...Object.getOwnPropertyDescriptors...

...hasOwnProperty

Own

↓

vlastnost může být vlastní nebo zděděná

↑

{a: 1, b: 2, c: 3}

≈

{a: 1} → {b: 2} → {c: 3}

```
// Klasicky object literal vytvari objekt,
// ktery dedi Object.prototype.
// Proto kazdy takovy objekt obsahuje metody
// jako `toString`, `valueOf` a `hasOwnProperty`.
const obj = {}
console.log(typeof obj.toString) // 'function'

// Pomoci staticke metody Object.create se da vytvorit skutecne
// prazdny objekt, ktery nededi nic.
// Takovy objekt neobsahuje zadne vlastnosti.
const empty = Object.create(null)
console.log(typeof empty.toString) // 'undefined'
```

```
// Vlastni vlastnosti patri primo danemu objektu.
const obj = {
    ownPropOne: 1,
    ownPropTwo: 2
}
console.log(
    Object.getOwnPropertyNames(obj) // ['ownPropOne', 'ownPropTwo']
)

// Zdedene vlastnosti jsou vlastni vlastnosti zdedenych objektu.
console.log(typeof obj.toString) // 'function'
console.log(
    Object.getOwnPropertyNames(obj.__proto__) // ['toString', 'valueOf', ...]
)
```

```
const o1 = {a: 1}
const o2 = {b: 2}
const o3 = {c: 3}

// Zjednodusene zapsana dedicnost.
o2.__proto__ = o1
o3.__proto__ = o2

console.log(
    o3.a, o3.b, o3.c
) // 1, 2, 3

console.log(
    Object.getOwnPropertyNames(o3)
) // 'c'
```

key: value

unikátní string nebo symbol

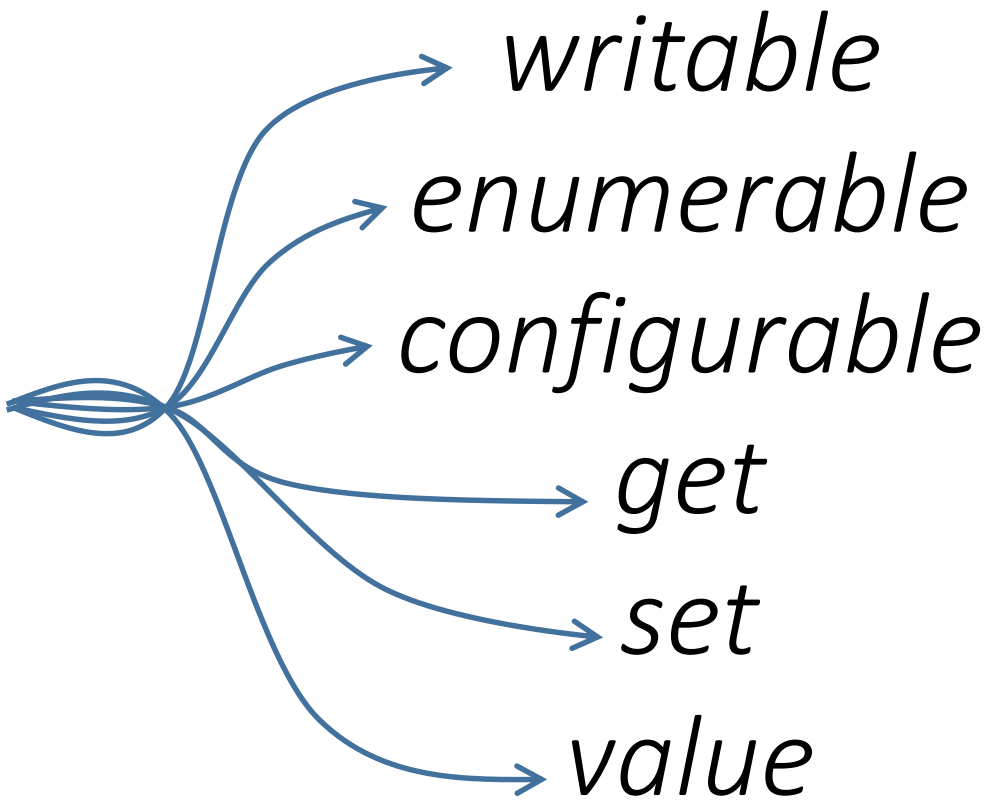
key: *value*

cokoliv

```
const obj = {a: 1}
// ... je stejny zapis jako ...
const obj = {'a': 1}
// ... je stejny zapis jako ...
const obj = {"a": 1}

// Vlastnost da se nadefinovat i pozdeji.
obj.newProp = 321
obj.anotherNewProp = 123

// Klice mohou byt jakekoliv stringy.
obj['this is name of property too'] = 777
console.log(obj['this is name of property too']) // 777

// Symboly.
const sym = Symbol('nejaky symbol')
const obj = {
    [sym]: 123
}
console.log(obj[sym]) // 123
```

descriptors

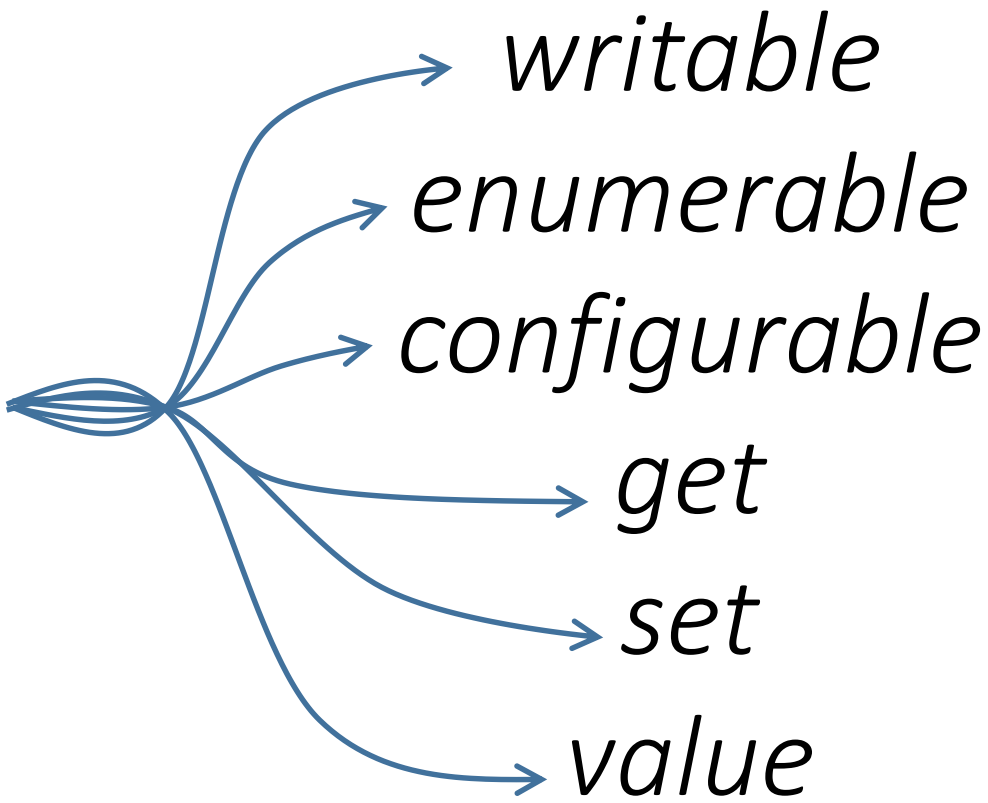
vlastnosti vlastnosti



descriptors

vlastnosti vlastnosti

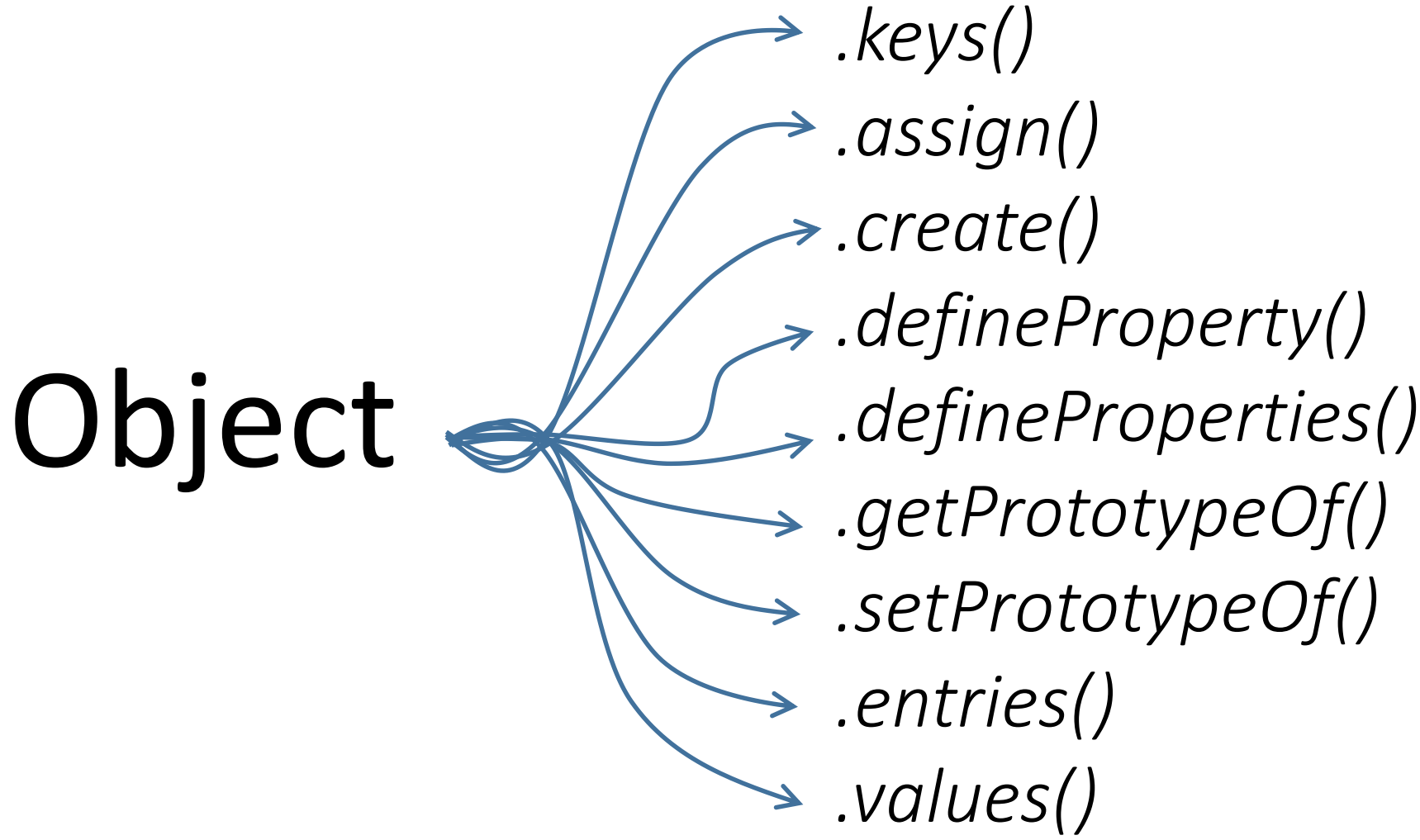
Object.defineProperties()
Object.defineProperty()
Object.create()



```
// Vlastnosti `a` a `b` jsou viditelne pro for cyklus.
const obj = {a: 1, b: 2}
for (const key in obj) console.log(key) // 'a', 'b'

// Vlastnost `b` neni pro cyklus for viditelna.
const obj2 = {a: 1, b: 2}
Object.defineProperty(obj2, 'b', {
    enumerable: false
})
for (const key in obj2) console.log(key) // 'a'
```

```
// Definovani deskriptoru pomoci Object.create()
const obj = Object.create(null, {
    a: {
        value: 123,
        configurable: true,
        writable: true,
        enumerable: false
    }
})
// Definovani deskriptoru pomoci Object.defineProperty()
const obj = {}
Object.defineProperty(obj, 'a', {
    enumerable: false
})
// Definovani deskriptoru pomoci Object.defineProperties()
const obj = {}
Object.defineProperties(obj, 'a', {
    a: {
        value: 123,
        enumerable: false
    }
})
```

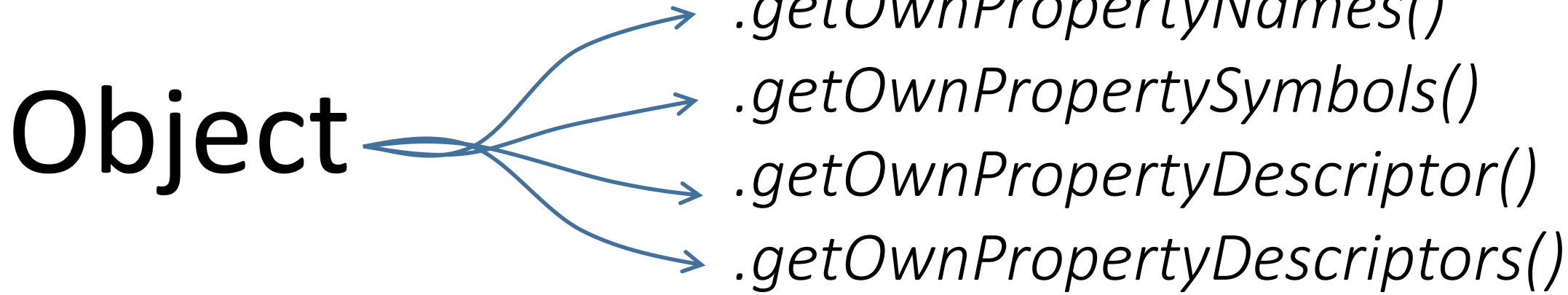
```
const obj = {
    filmy: ['Forest Gump', 'Avengers', 'Home Alone']
}
Object.defineProperty(obj, 'length', {
    get: function() {
        return obj.filmy.length
    },
    set: function(len) {
        obj.filmy.length = len
    }
})

console.log(obj.length) // 3
obj.length = 2
console.log(obj.filmy) // ['Forest Gump', 'Avengers']
console.log(obj.length) // 2
console.log(obj.filmy.length) // 2
```


Object
.keys()
.assign()
.create()
.defineProperty()
.defineProperties()
.getPrototypeOf()
.setPrototypeOf()
.entries()
.values()

```
const obj = {a: 1, b: 2, c: 3}

// Object.keys - vraci pole vlastnich klicu.
console.log(Object.keys(obj)) // ['a', 'b', 'c']

// Object.assign - prirazuje vlastnosti vsech danych objektu.
// (od posledniho argumentu k prvnimu)
const obj2 = Object.assign({d: 4}, obj, {a: 777}) // {a: 777, b: 2, c: 3, d: 4}

// Vraci prototype object (stejny jako __proto__).
Object.getPrototypeOf([]) === Array.prototype
Object.getPrototypeOf([]) === [].__proto__

// Nastavuje prototype object (bad practice).
const arr = []
Object.setPrototypeOf(arr, {}) // stejne jako arr.__proto__ = {}
console.log(typeof arr.map) // 'undefined'
```

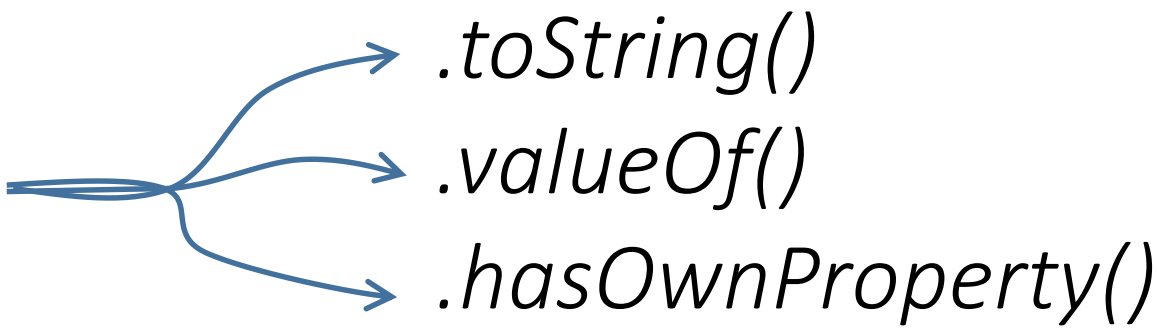
p.3 → *Object jako datová struktura*

```
const obj = {a: 1, b: 2, c: 3}

console.log(Object.entries(obj)) // [ [ 'a', 1 ], [ 'b', 2 ] ]
console.log(Object.values(obj)) // [ 1, 2 ]
```


Object
.getOwnPropertyNames()
.getOwnPropertySymbols()
.getOwnPropertyDescriptor()
.getOwnPropertyDescriptors()

```
const obj = {a: 1}
const sym = Symbol('test symbol')
Object.defineProperty(obj, 'b', {
    enumerable: false,
    value: 2
})
obj[sym] = 3

console.log(obj.a, obj.b, obj[sym]) // 1, 2, 3

console.log(Object.getOwnPropertyNames(obj)) // [ 'a', 'b' ]
console.log(Object.getOwnPropertySymbols(obj)) // [ sym ]
console.log(Object.getOwnPropertyDescriptors(obj)) // {a: ..., b: ..., ...}
console.log(Object.getOwnPropertyDescriptor(obj, 'b')) // {enumerable: false, ...}
```

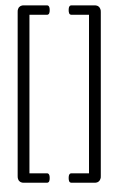
Object.*prototype*



```
const obj = {a: 1}
console.log(obj.toString()) // '[object Object]'
console.log(obj.valueOf()) // {a: 1}
console.log(obj.hasOwnProperty('a')) // true

const inner = {b: 2}
obj.__proto__ = inner
console.log(obj.hasOwnProperty('b')) // false
console.log(obj.b) // 2
```

Array

p.3 → *Pole jako datová struktura*



[1, 2, 3]

[*1*, *2*, *3*]

array

kontejner prvků v určitém pořadí

p.3 → *Pole jako datová struktura*

0 … n

↓

index prvku, délka pole

↑

počet prvků

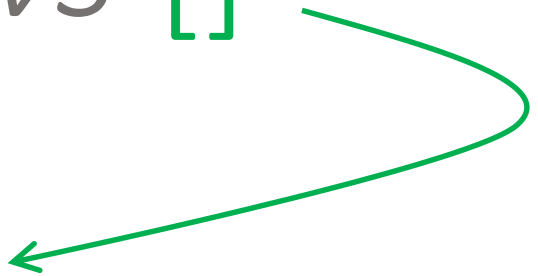
```
const arr = ['raz', 'dva', 'tri']

console.log(arr[1]) // 'dva'
console.log(arr.length) // 3
```

typeof [] === ‘object’

```
const arr = ['raz', 'dva', 'tri']

console.log(typeof arr) // 'object'
console.log(arr.constructor) // Array
console.log(arr instanceof Array) // true
console.log(Object.prototype.toString.call(arr)) // '[object Array]'
```

Array() *vs* new Array() *vs* []

Array() *vs* new Array() *vs* []



nejrychlejší a nejbezpečnější

```
console.log(new Array(1, 2, 3)) // [1, 2, 3]
console.log(Array(1, 2, 3)) // [1, 2, 3]
console.log([1, 2, 3]) // [1, 2, 3]

console.log(Array(5)) // [ <5 empty items> ]
```

```
[1, 2, 3] !== [1, 2, 3]
```

```
const arr1 = [1, 2, 3]
const arr2 = [1, 2, 3]

console.log(arr1 === arr2) // false
console.log(arr1 === [1, 2, 3]) // false
console.log(arr1 === arr1) // true
```

Pseudopole

array-like object

arguments, NodeList, HTMLCollection

p.3 → *Pole jako datová struktura*

```
function x() {
    console.log(arguments) // { '0': 123 }
    console.log(typeof arguments.map) // 'undefined'
    console.log(arguments instanceof Array) // false
}

x(123)
```

Iterování

všechny cykly kromě for…in

p.3 → *Pole jako datová struktura*

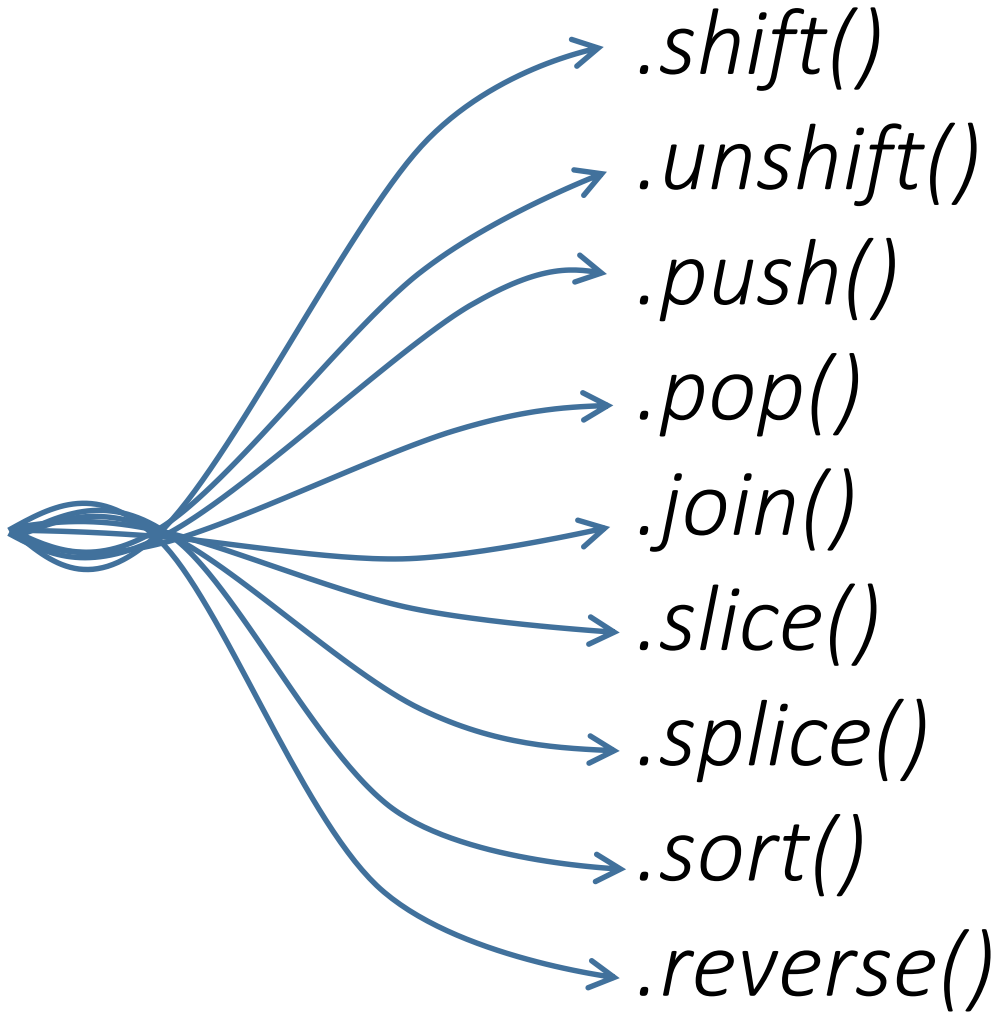
```
const arr = ['raz', 'dva', 'tri']

// for ;;
for (let i = 0; i < arr.length; i++) console.log(i, arr[i])
// for ;; s kesovanim delky
for (let i = 0, len = arr.length; i < len; i++) console.log(i, arr[i])
// while
let i = 0; while(i < arr.length) console.log(i, arr[i++])
// for..of
for (const value of arr) console.log(value)
// for..in <-- nepouzivat
for (const key in arr) console.log(key, arr[key])
```

p.3 → *Pole jako datová struktura*

vlastnost .length

```
const arr = ['raz', 'dva', 'tri']
console.log(arr.length) // 3

const arr2 = Array(5)
console.log(arr2.length) // 5

const arr3 = []
arr3[999] = null
console.log(arr3.length) // 1000

const arr4 = ['raz', 'dva', 'tri']
arr4.length = 5
console.log(arr4) // ['raz', 'dva', 'tri', <2 empty items>]

const arr5 = ['raz', 'dva', 'tri']
arr5.length = 2
console.log(arr5) // ['raz', 'dva']
```

Array.*prototype*



```
let arr = ['raz', 'dva', 'tri']
console.log(arr.shift()) // 'raz'
console.log(arr) // ['dva', 'tri']

arr = ['raz', 'dva', 'tri']
console.log(arr.unshift('ahoj')) // 4
console.log(arr) // ['ahoj', 'raz', 'dva', 'tri']

arr = ['raz', 'dva', 'tri']
console.log(arr.push('neco na konec')) // 4
console.log(arr) // ['raz', 'dva', 'tri', 'neco na konec']

arr = ['raz', 'dva', 'tri']
console.log(arr.pop()) // 'tri'
console.log(arr) // ['raz', 'dva']
```

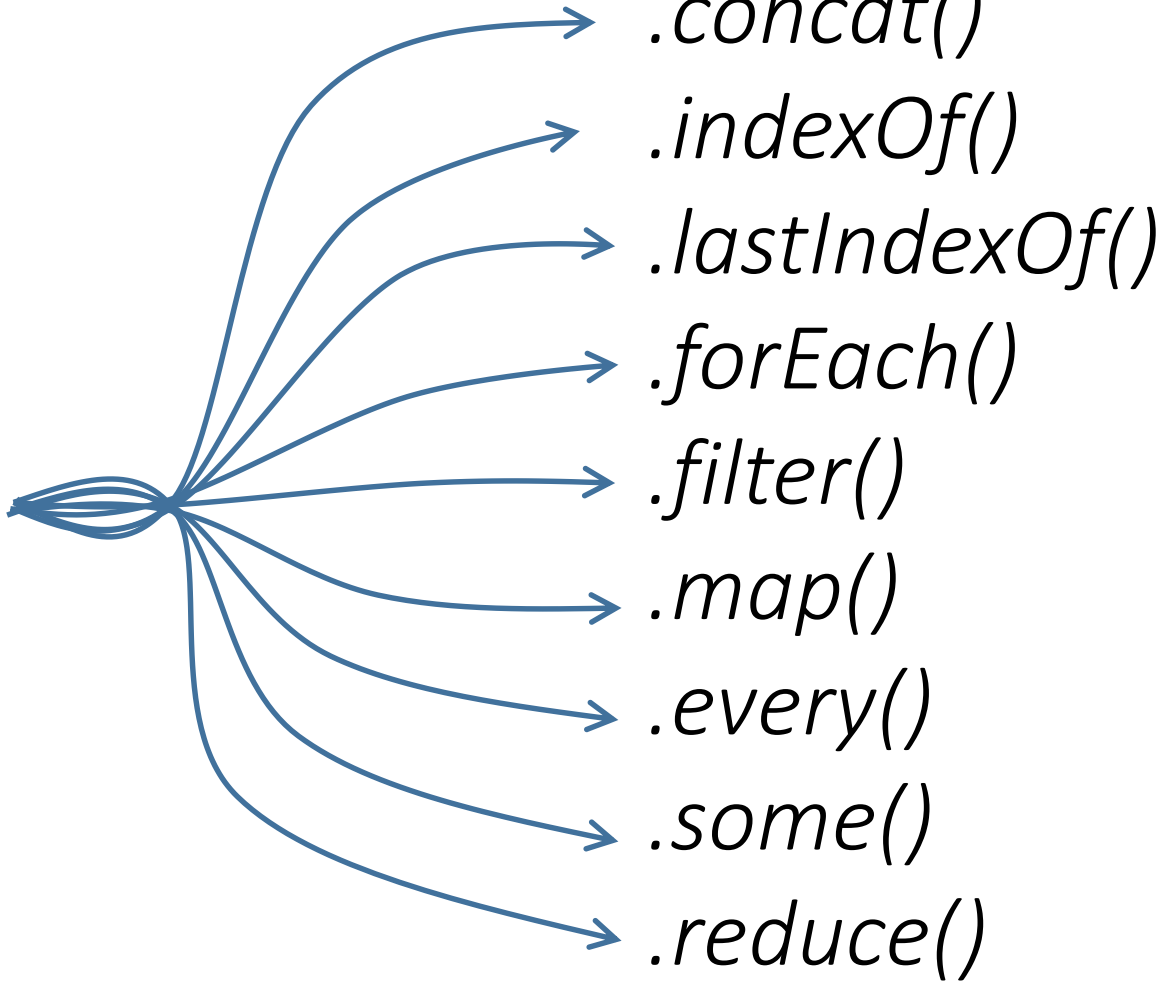
```
const arr = ['raz', 'dva', 'tri']
console.log(arr.join(' a potom ')) // 'raz a potom dva a potom tri'

console.log(arr.slice()) // ['raz', 'dva', 'tri']
console.log(arr.slice(1)) // ['dva', 'tri']
console.log(arr.slice(1, -1)) // ['dva']
console.log(arr.slice(0, 2)) // ['raz', 'dva']
```

```
const arr = ['raz', 'dva', 'tri']

arr.splice(2, null, 'dva a pul') // []
console.log(arr) // ['raz', 'dva', 'dva a pul', 'tri']

arr.splice(1, 2) // ['dva', 'dva a pul']
console.log(arr) // ['raz', 'tri']

arr.splice(0, 1, 'dva', 'dva a pul') // ['raz']
console.log(arr) // ['dva', 'dva a pul', 'tri']

const clone = arr.slice()
clone.splice(2, 1, 'ctyri') // ['tri']
console.log(clone) // ['dva', 'dva a pul', 'ctyri']
console.log(arr) // ['dva', 'dva a pul', 'tri']
```

```
const arr = [3, 1, 2]
console.log(arr.sort()) // [1, 2, 3]
console.log(arr.sort(function(a, b) {return b - a})) // [3, 2, 1]
console.log(arr) // [3, 2, 1]

const arr2 = ['raz', 'dva', 'tri']
console.log(arr2.reverse()) // ['tri', 'dva', 'raz']
console.log(arr2) // ['tri', 'dva', 'raz']
```

p.3 → *Pole jako datová struktura*

Array.*prototype*



p.3 → *Pole jako datová struktura*

```
const arr1 = ['a', 'b']
const arr2 = ['c', 'd', ['e', 'f']]
console.log(arr1.concat(arr2)) // ['a', 'b', 'c', 'd', ['e', 'f']]

const arr = [1, {a: 123}, 'tri']
console.log(arr.indexOf('tri')) // 2
console.log(arr.indexOf({a: 123})) // -1

const obj = {a: 123}
const arr3 = [1, obj, 'tri']
console.log(arr3.indexOf(obj)) // 1
```

```
const arr = [1, 2, 'tri']

const numbersOnly = arr.filter(function(value) {
    return typeof value === 'number'
}) // [1, 2]

const evenOnly = arr.filter(function(value, index) {
    return index % 2 ? true : false
}) // [2]

console.log(arr) // [1, 2, 'tri']
```

```
const arr = [1, 2, 3]

const sqrts = arr.map(function(value) {
    return Math.sqrt(value)
}) // [ 1, 1.4142135623730951, 1.7320508075688772 ]
```

```
const arr = [1, 2, 3]

arr.forEach(function(value, index) {
    console.log(value, index)
}) // 1 0, 2 1, 3 2

console.log(arr.some(function(value) {
    return value > 3
})) // false

console.log(arr.every(function(value) {
    return typeof value === 'number'
})) // true
```

```
const arr = [1, 2, 3]

const sum1 = arr.reduce(function(prev, curr) {
    return prev + curr
}, 0) // 6

const sum2 = arr.reduce(function(prev, curr) {
    return prev + curr
}) // 6
```

Úkoly → bit.ly/2IX20Jh

// end