Bonus: Funkcionální programování v JavaScriptu

ES6/ES2015

- nová „šipková“ syntaxe pro funkce:
  (*parametry*) => { *tělo* }
- je-li jen 1 parametr a tělem samotný výraz, lze vynechat závorky a return
- původně:
  function mocnina (x) { return x * x; }
- nyní lze:
  const mocnina = (x) => { return x * x; }
  const mocnina = x => x * x;

Funkce jako parametr

```
const hledej = ( porovnani, co, kde ) => {
    for ( let x of kde ) {
        if ( porovnani( x, co ) ) return true;
    }
    return false;
}

console.log( hledej( (x,y) => x===y, 3, [1, 2, 3, 4] ) );
```

Funkce jako výsledek

```
const intervalGen = ( dolni, horni ) =>
    x => dolni <= x && x <= horni;

const maleCislo = intervalGen( 0, 10 );

console.log( maleCislo( 5 ) );      → true
console.log( maleCislo( 15 ) );     → false
```

Map

- metoda polí
- dostane funkci, kterou zavolá pro každý prvek pole a vytvoří pole z výsledků
- const data = [1, 2, 3, 4, 5];
  const mocniny = data.**map(x => x*x)**;
  [1, 4, 9, 16, 25]
- funkce může mít až 3 parametry:
  aktuální hodnota, index, původní pole

Filter

- parametrem pravdivostní funkce, výsledkem pole těch hodnot z původního pole, pro které vyšlo true
- const liche = data.**filter(x => x%2)**;

  [1, 3, 5]
- lze řetězit:

  const liche2 = data.filter(x => x%2).map(x => x*x);

  [1, 9, 25]

Reduce

- výpočet hodnoty z pole, parametry:
  - funkce se 2 parametry: akumulátor a hodnota
  - inicializační hodnota akumulátoru
- inicializuje akumulátor; pro každou hodnotu v poli zavolá funkci a nahradí akumulátor jejím výsledkem
- const soucet = data.**reduce((suma,x) => suma+x, 0)**;

  15

  const soucin = data.**reduce((mul,x) => mul*x, 1)**;

  120

Funkcionální JavaScript

- má všechny klíčové vlastnosti:
  - funkce 1. třídy – funkce je hodnota jako každá jiná
  - anonymní funkce/funkcionální výrazy – x => x*x
  - uzávěry – funkce vrácená jako výsledek si uchová své lokální prostředí
- poslední dobou získává značnou popularitu