Distribueret programmering:

Lektion 02: Funktioner

Funktioner

- En funktion har
  - nul, en eller flere parametre
  - en block ({}) med sætninger og evt. erklæring af lokale variable og lokale funktion(er)
- Funktioner er *first-class elementer* i JavaScript, dvs. funktioner kan være værdien af variable, konstanter og parameter, samt returneres af funktioner
- Parametre har ligesom variable og konstanter ingen type
- Der er ingen tjek af det antal argumenter, en funktion kaldes med
- Manglende argumenter får værdien undefined

Funktioner kan erklæres på 3 måder

- Navngivet funktion
- Anonym funktion
- Arrow funktion

```
// funktioner.js
function f(i) { return 2 * i; }
console.log(f(1)); // => 2

const g = function(i) { return 2 * i; };
console.log(g(2)); // => 4

const h = i => 2 * i;
console.log(h(3)); // => 6

const h0 = () => 8;
console.log(h0()); // => 8

const h2 = (i,j) => { return i * j; };
console.log(h2(2,5)); // => 10
```

return sætning

- En return sætning afslutter funktionen og returnerer en værdi
- Hvis der ikke er angivet en return værdi – eller return sætningen mangler, returneres værdien undefined

```
// return.js
function f() {}
console.log(f()); // => undefined

function g(i) { if (i < 0) return; else return i; }
console.log(g(-1)); // => undefined
console.log(g(2)); // => 2
```

Lokale funktioner

- Funktioner kan have lokale funktioner
- Bemærk at navngivne funktioner hoistes til starten af en funktion/
  et program. Tænk på g som en erklæret funktion og h som en variabel, der er undefined indtil den gives en værdi.

```
// lokale.js
f();
function f() {
    console.log(g()); // => 1
    function g() { return 1; }
    console.log(h()); // => ReferenceError: Cannot access 'h' before initialization
    const h = function() { return 2; };
}
```

scope

- Variable og konstanter erklæret udenfor funktioner har global scope
- Parametre samt lokale variable erklæret med var har function scope
- Lokale navngivne funktioner har også function scope
- Variable erklæret med let og const har block scope

```
// scope.js
function f(max) {
    let sum = 0;
    for (let i = 1; i <= max; i++) { sum += i; }
    console.log(max); // => 10
    console.log(sum); // => 55
    console.log(i); // => ReferenceError: i is not defined
}
f(10);
```

Default værdier

- Parametre (ofte de sidste) til en funktion kan gives en default værdi

```
// default.js
function f(a, b = '.') {
    return a + b;
}
console.log(f(1, 2)); // => 3
console.log(f(3)); // => 3

function megetLang(p1 = 'a ', p2, p3, p4, p5, p6 = ' f') {
    return p1 + p2 + p3 + p4 + p5 + p6
}
console.log(megetLang('a', 'b', 'c', 'd'))  // abcdundefined f
console.log(megetLang()) // a undefinedundefinedundefinedundefined f
```

Closure

- En lokal funktion har en kopi (kaldet en closure) af sin omgivende funktions variable og parametre, når den returneres eller videregives
- Dermed har hver ekstern udgave af en lokale funktion sin egen private hukommelse

```
// closure.js
function next() {
    let n = 1;
    return function(){return n++;};
}
const next1 = next();
const next2 = next();
console.log(next1()); // => 1
console.log(next1()); // => 2
console.log(next2()); // => 1
```

Callback

- Closure kan fx udnyttes i callbacks – hvor en lokal funktion er et argument til en asynkron funktion
- Vi vender tilbage til dette senere

```
// callback.js
function afvent(v) {
    function callback(){console.log(v);}
    setTimeout(callback, 1000);
    console.log('afvent ...');
}
afvent(1); // => afvent ...
afvent(2); // => afvent ...
// => 1
// => 2
```