INFO134 Klientprogrammering

Ukesoppgaver

Uke 5

1 Funksjoner og variabler 1

For hver av kodesegmentene under, analyser koden og vurder følgende:

1. hvilke navn blir definert i ytterste nivå (funksjoner og variabler)?
2. dersom funksjonene `f`, `f0`, `f1`, `f2`, `f3` er definerte, hva returnerer de hvis de kalles uten argumenter?
3. hvilke av funksjonene som finnes er *identiske?* (`f === f0`? `f0 === f1`?)

Ta stilling til spørsmålene før du skriver inn koden og tester.

1.a)

```
1         var i = 0;
2         while(i < 3) {
3           function f() {
4             return i;
5           }
6           window["f" + i] = f;
7           i++;
8         }
9         i = 0;
```

Svar: variabelen i er definert og har verdi 0. funksjonene f, f0, f1, f2 er definert. alle returnerer 0 (verdien av i).

1.b)

```
1         function example(){
2           var i = 0;
3           while(i < 3) {
4             function f() {
5               return i;
6             }
7             window["f" + i] = f;
8             i++;
9           }
10          i = 0;
11        }
12
13        example();
```

Svar: Funksjonene f0, f1, f2, f3 er alle definert (ytterst). De er egenskaper av det globale objektet. Alle funksjonene returnerer 0 som er verdien av variabelen i *i funksjonen!* (Husk at funksjoner har referanse til miljøet de ble opprettet i!)

1.c)

```
function example(){
  i = 0;
  while(i < 3) {
    function f() {
      return i;
    }
    window["f" + i] = f;
    i++;
  }
  i = 0;
}

example();
```

Svar: Samme som 1.b), men nå er også variabelen i en global variabel. Nå har vi ikke `var` foran som gjør det til en lokal variabel, men det blir en global variabel i stedet!

1.d)

```
function example(){
  i = 0;
  while(i < 3) {
    f = function() {
      return i;
    }
    window["f" + i] = f;
    i++;
  }
  i = 0;
}

example();
```

Svar: Samme som forrige oppgave, men nå er både i og f definert som globale funskjoner. Vi har altså funksjonene: f, f0, f1, f2. Alle funksjonene returnerer 0 (verdien av i).

1.e)

```
  function example(){
    i = 0;
    while(i < 3) {
      var f = function() {
        return i;
      }
      window["f" + i] = f;
      i++;
    }
    var i = 0;
  }

  example();
```

Svar: Som i forrige oppgave, men nå er i en *lokal* variabel fordi vi erklærer den på slutten av funksjoneskroppen!

1.f)

```
  function example(){
    i = 0;
    while(i < 3) {
      f0 = function() {
        return i;
      }
```

```
7                 window["f" + i] = f0;
8                 i++;
9               }
10              var i = 0;
11            }
12
13            example();
```

Svar: Som tidligere. Funksjonene f0, f1, f2 er definert (i er ikke global). f0 er identisk med f2, men de er ikke identisk til f1!

1.g)

```
1             function makeF(i) {
2               function f0() {
3                 return i;
4               }
5               return f0;
6             }
7
8             function example(){
9               i = 0;
10              while(i < 3) {
11                window["f" + i] = makeF(i);
12                i++;
13              }
14              var i = 0;
15            }
16
17            example();
```

Svar: Merk at funksjonene vi oppretter har en referanse til `makeF`-*kallet* som opprettet dem. I hver av dem har variabelen i en egen verdi. f0, f1, f2 er definert og de returnerer henholdsvis 0, 1, og 2.

1.h)

```
1               function example(){
2                 i = 0;
3                 while(i < 3) {
4                   (function(i){
5                     function f() {
6                       return i;
7                     }
8                     window["f" + i] = f;
9                   })(i);
10                  i++;
11                }
12                var i = 0;
13              }
14
15              example();
```

Svar: Nøyaktig som forrige oppgave, men vi har erstattet `makeF`-funksjonen med en anonym funksjon! (Merk at `f` er *ikke* definert – denne er bare definert inni den anonyme funksjonen.)

1.i)

```
1               function example(){
2                 i = 0;
3                 while(i < 3) {
4                   (function(i){
5                     f0 = function() {
6                       return i;
```

```
7                     }
8                     window["f" + i] = f0;
9                   })(i);
10                  i++;
11                }
12                var i = 0;
13              }
14
15              example();
```

Svar: f0, f1, f2 er nå definert. men merk at vi oppretter funksjon til en variabel, f0, som ikke er erklært som en lokal variabel! f0 og f2 er samme objekt: `f0 === f2`, og begge returnerer 2 (siste definisjon av f0, og så tilordnet til f2.)

2 Variabler, objekter, funksjoner: samme navn

Opprett tre filer: `person.js`, `car.js`, og `program.js`. Programmet er enkelt:

```
1   // skriv ut personen
2   var p = constructPerson("Per", "Nielsen");
3   console.log(p.name());
4
5   // skriv ut verbet
6   console.log("... owns...");
7
8   // skriv ut bilen
9   var c = constructCar("ST", 12345);
10  console.log(c.reg());
```

I filen `person.js` skal du definere et objekt, `methods`, som definerer én metode: `name`. Du skal også definere én funksjon som oppretter et person-objekt som har fornavn, etternavn, og metoden `name`.

```
1   function constructPerson(first, last) {
2     var newObject = Object.create(methods);
3     newObject.first = first;
4     newObject.last = last;
5     return newObject;
6   }
7
8   var methods = {
9     name: function() {
10      return this.first + " " + this.last;
11    }
12  }
```

I filen `car.js` skal du definere en funksjon, `constructCar` som tar to argumenter: bokstaver (f. eks. "ST") og tall (f. eks. 12345). Du skal også definere objektet `methods` som har metoden `reg` som kobinerer og returnerer disse to verdiene.

For hver av tilfellene under, finn ut hva som vil skje før du skriver koden og implementerer.

2.a) Definer et HTML-dokument som importerer, i denne rekkefølgen:

1. `person.js`
2. `program.js`

Svar: Skriver ut person-info og tekstverdien, men krasjer når vi forsøker å opprette et bil-objekt.

2.b) Importer i rekkefølge:

1. `car.js`
2. `program.js`

Svar: Krasjer umiddelbart fordi første funksjonskall (constructPerson) er til en udefinert funksjon.

2.c) Importer i rekkefølge:

1. `person.js`
2. `car.js`
3. `program.js` **Svar:** Nå er begge funksjonene definert, men "`p.name` is not a function". Det som har skjedd er at når vi kalte `constructPerson` så har `methods`-variabelen blitt tilordnet et objekt som bare har én egenskap: `reg`. Når vi kalte `constructPerson`-funksjonen går alle stegene igjennom og vi får ut et resultat, men objektet som returneres har metoden `reg`, og ikke metoden `name`!

2.d) Importer i rekkefølge:

1. `car.js`
2. `person.js`
3. `person.js`

Svar: Som i forrige oppgave, men nå er det `car`-objektet som ikke har metoden sin (`reg`).

2.e) Endre `person.js` og `car.js` slik at du ikke har et `methods`-objekt for hver av dem, men at i begge tilfeller arver `construcXYZ`-funksjonen fra et objekt som er en egenskap av den funksjonen kalt `prototype`. Altså, `constructPerson`-funksjonen skal opprette et objekt (vha. `Object.create`) som arver fra `constructPerson.prototype`, og tilsvarende for `constructCar`-funksjonen.

2.f) Kopier og skriv om `person.js` og `car.js` til å bruke konstruktører. Kall konstruktøren `PersonA`. Bruk følgende mønster:

```
function PersonA(first, last) {
  ⋮
}

PersonA.prototype = {
  name: function() {
    ⋮
  }
};
```

2.g) Skriv om filene igjen og bruk følgende mønster (kall konstruktøren `PersonB`):

```
function PersonB(first, last) {
  ⋮
}

PersonB.prototype.name = function() {
    ⋮
};
```

2.h) La `p = new PersonA("Per", "Nielsen");` eller `p = new PersonB("Per", "Nielsen");` For hvilken av disse to er følgende rett? Hvorfor?

- `p.constructor === PersonA`, eller
- `p.constructor === PersonB`?

Svar: Dette fungerer for `PersonB` fordi vi ikke overskriver `prototype`-egenskapen, men bare legger til en ny egenskap (metoden `name`). Funksjonen `PersonB` har allerede `prototype`-egenskap med riktig definert `constructor`-egenskap.

3 Lett å rote med variabler

3.a) Implementer Example 13-1 (men del det gjerne i ulike filer). Hvordan fungerer klikking for å vise/skjule elementer?

3.b) Legg til et nytt `div`-element som skal skjule informasjon og vise det når du klikker på "handle"-elementet. Hvordan forventer du at dette skal fungere? Hvordan fungerer det?

3.c) Skriv inn funksjonen under og bruk den til å forenkle eksemplet du nettopp implementerte. HVordan fungerer klikking nå?

```
function makeReveal(elt) {
  var title = elt.getElementsByClassName("handle")[0];
  // When that element is clicked, reveal the rest of the content
  title.onclick = function() {
    if (elt.className == "reveal") {
      elt.className = "revealed";
    } else if (elt.className == "revealed") {
      elt.className = "reveal";
    }
  };
}
```

Hvordan fungerer klikking nå? Hvorfor?