INFO134 Klientprogrammering

Ukesoppgaver

Uke 3

1 Grunnleggende

JavaScript har to "konstanter" som representerer "fravær av verdi": `undefined` og `null`.

I Python brukte vi hovedsakelig `None` for å representere dette. Returverdien av en funksjon som ikke har en `return`-setning (eller en `return` uten argument), er `None`. Vi kunne også bruke `None` hvis vi vil opprette en variabel, men ikke enda visste hvilken verdi den skulle ha (`x = None`).

I JavaScript har vi altså to (les *3.4 null and undefined* (s. 41, 42).)

JavaScript har også to former for likhetstesting: `==` og `===`. Den første formen er en "likhetstest med typekonversjon". Det betyr at likhetstesten evaluerer til sann hvis de to verdiene (uttrykkene) vi sammenligner kan konverteres til samme type, og *da* ha lik verdi. I oppgavene under skal du svare på noen spørsmål om JavaScript. Tenk likevel gjerne over om det er ==-testen eller ===-testen i JavaScript som fungerer mest som ==-testen i Python.

1.a) Er tekstverdien `"1"` lik tallet `1`? Åpne konsollen og skriv inn: `"1" == 1;` og `"1" === 1`.

1.b) Tilsvarer de boolske verdiene, `true` og `false`, tallene `0` og `1`?

1.c) Er `null` og `undefined` like?

1.d) I matematikken er ikke brøken $\frac{1}{0}$ meningsfull, det tilsvarer ikke et tall. Hva er verdien av `1 / 0`?

1.e) Er `Infinity == undefined`?

1.f) Er `1/0` det samme som `1/-0`?

1.g) Hva er `1 + "4"`? Hva er `1 + ("4" / 2)`?

1.h) Flyttall kan ofte føre til feil som er vanskelig å finne. Det er lurt å aldri sammenligne flyttall direkte. Definer variablene `var x = 0.3 - 0.2;` og `var y = 0.2 - 0.1;`. Er disse tallene like? Test `x == y`. (Dette gjelder de aller fleste programmeringsspråk; se 3.1.4 *Binary Floating-Point and Rounding Errors* s. 34, 35 for mer informasjon.)

1.i) JavaScript aksepterer veldig mange uttrykk som "boolske" uttrykk (uttrykk av sannhetsverdier). Definer følgende funksjon:

```
function ifTest(x) {
  if (x) {
    return x + " er 'truthy'.";
  } else {
    return x + " er 'falsy'.";
  }
}
```

Hvilke *tall* tror du JavaScript vil anse som "truthy" og "falsy" (du kan lese om dette på s. 40, 3.3 *Boolean Values*)? Kall `ifTest(-2);`, `ifTest(-1);`, `ifTest(-0);`, `ifTest(0);`, `ifTest(1);`, `ifTest(2);`.

1.j) Vi kan bruke dette av JavaScript til å gjøre noe *bare dersom* noe er definert, og eller gjøre noe annet (eller ingenting). Hva tror du du får når du gjør `ifTest(null);` og `ifTest(undefined);`? Hva med `ifTest({});` eller `ifTest({}.enEgenskap);`?

1.k) Pythons intuitive navn for boolske konnektiver finnes ikke i JavaScript. I JavaScript heter negasjon ("`not`") `!`, konjunksjon ("`and`") heter `&&`, og disjunksjon ("`or`") heter `||`. Sjekk at "og" og "ikke" fungerer som forventet. Hva tror du følgende vil evaluere til?

- `!(false && false);`?
- `!(false && true);`?
- `!(true && false);`?
- `!(true && true);`?

1.l) Tror du `!0 == true;`? Hva med `!0 === true;`?

1.m) Hva tror du `!null || {};` evaluerer til? Test etter at du har formulert et svar. Hvorfor blir det slik?

1.n) Hva blir `!(null || {});` og `null || !{};`?

2 Objekter representerer fenomener

I denne oppgaven skal du skrive JavaScript-programmer som vi laster inn fra et ellers ganske tomt HTML5-dokument.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Laboppgave</title>
    <meta charset="utf-8">
    <script type="text/javascript" src="oppg1_a.js"></script>
  </head>
  <body>
  </body>
</html>
```

I disse oppgavene skal vi bare skrive ut informasjon i konsollen. Derfor er det ikke så viktig å strukturere JavaScript-programmet. Vi kan la alt evalueres og kjøres direkte uten å vente på "`load`"-hendelsen til `body`-elementet, eller lignende.

2.a) Definer et objekt som representerer et konkret registreringsnummer for bil. Objektet skal følgende to egenskaper (og verdi): `bokstaver` skal være "EL", og `tall` skal være 12345. Tilordne dette objektet til variabelen `o1` (husk nøkkelordet `var`).

2.b) Logg `o1` og undersøk at du har definert objektet du skal.

```
» console.log(o1);
  ▶ Object { bokstaver: "EL", tall: 12345 }
← undefined
```

```
» o1;
← ▶ Object { bokstaver: "EL", tall: 12345 }
```

2.c) Vi kan få lese ut eller skrive over verdien av en egenskap enten ved punktum-notasjon, eller ved å angi en tekstverdi i firkantparenteser. Endre `bokstaver`-egenskapen til "ST" ved ved hjelp av punktum-notasjon, og `tall`-egenskapen til 23456 ved hjelp av firkantparenteser. Les ut verdiene med begge teknikkene.

2.d) Definer et nytt objekt og tilordne det til en ny variable, `o2`. Objektet skal ha de samme egenskapene som `o1`, men skal også ha en *getter* for egenskapen `regNr`. Sjekk at du får samme oppførsel som vist under. Hva skjer dersom du tilordner `regNr` en ny verdi? Hva skjer med `regNr`-egenskapen dersom du endrer en av de andre egenskapene?

2.e) Opprett et objekt, `p`, som representerer en person. La objektet ha egenskapene fornavn og etternavn, samt en `getter` som heter `navn` som setter sammen de andre egensapene som resultat.

2.f) Opprett en variabel `fam` som er en "liste" med tre/fire person objekter som over. Skriv ut navnene på familiemedlemmene med `for`-løkken:

```
1  for(person in fam) {
2    console.log(person.navn);
3  }
```

(I JavaScript heter ikke denne datatypen "list", men "Array". `append`-metoden vi kjenner fra Python heter `push` nå. Vi lærer mer om det senere.)

3 Arv

3.a) Vi ønsker å redusere repetisjonen vi hadde i forrige oppgave. Definer objektet `protoPerson` som *bare* har `getter`-en du definerte for hver person. Sjekk at når du leser verdien av `protoPerson.navn`, så får du tekstverdien "undefined undefined". Hva skjer?

3.b) Opprett objektet `var ola = Object.create(protpPerson);` og gi objektet fornavn- og etternavn-egenskaper hhv. "Ola" og "Nielsen". Les ut verdiene: "fornavn", "etternavn" og "navn".

3.c) Fullfør programmet under. Programmet skal skrive ut fire linjer med tekst: Ole Nielsen, Kari Nielsen, Mor Nielsen, og Far Nielsen. (Strengt talt kan rekkefølgen variere.)

```
1  var protoPerson = // fra 3.a)
2
3  function person(fornavn, etternavn) {
4    // opprett objekt som har protoPerson som prototype
5    // ved hjelp av Object.create-metoden
6    // gi deretter objektet egenskaper 'fornavn' og 'etternavn'
7    // returner objektet
8  }
9
10 var familie = {
11     ola: person("Ola", "Nielsen"),
12     kari: person("Kari", "Nielsen"),
13     mor: person("Mor", "Nielsen"),
14     far: person("Far", "Nielsen")
15 };
16
17 for (var nøkkel in familie) {
18     var verdi = familie[nøkkel];
19     console.log(verdi.navn);
20 }
```

3.d) Tenk deg at du utvider programmet fra forrige oppgave med følgende programsetning:

```
protoPerson.hils = function() { console.log("Hei, " + this.navn + "!"); }
```

- Hypotetisk: Hva har du gjort nå?
- Hypotetisk: Hva skjer når du skriver `familie.ola.hils();` i konsollen?
- Hypotetisk: Hvor mange funksjoner/metoder ble aktivert når du skrev instruksen i konsollen?

Skriv inn utvidelsen. Stemte forventningene dine?

3.e) Hvilken datastruktur/teknikk ville du ha brukt i Python for å beskrive (dersom det er mulig):

- variabelen `familie`?

- variabelen `ola` dersom vi lar `var ola = familie["ola"];`)?
- variabelen `protoPerson`?

3.f) I tillegg til forrige oppgave (som jeg antar heter `oppg3_c.js`), skal du nå skrive et nytt program som du skal laste inn *etter* forrige:

```
...
<head>
...
<script type="text/javascript" src="oppg3_c.js"></script>
<script type="text/javascript" src="oppg3_f.js"></script>
...
```

Opprett et objekt, `protoSpesPerson`, som har `protoPerson` som prototype. La `protoSpesPerson` ha følgende egenskaper:

- `protoSpesPerson.alder = function() { return 2018 - this.født; }` (Folk flest har enda ikke hatt bursdag i år...)
- Metoden `hils` som skal skrive ut ("logge") teksten "Ærede ..." etterfulgt at navnet til den spesielle personen.

Opprett objektet `var elizabeth` med fornavn "Elizabeth", etternavn "II", og egenskapen `født` satt til 1926. Hvor gammel er Elizabeth? (Hun har ikke bursdag før i april.) Både `elizabeth` og `ola` er personer (arver fra `protoPerson`): hvordan hilser programmet ditt på dem?

4 Tilstand og interaksjon

4.a) Lag et tomt HTML5-dokument som importerer (f.eks.) `oppg4_a.js`. I denne JavaScript-filen skal du opprette et objekt `var teller`.

Dette objektet skal ha en egenskap `counter` med verdi 3. Objektet skal også ha en metode `tick` som gjør følgende:

- Først sjekker vi om `counter`-egenskapen er null. I så fall returnerer vi fra metoden.
- Minsker `counter`-egenskapen med én.

Last inn dokumentet og kall `tick`-metoden til den returnerer `undefined`. Overbevis deg selv (og/eller en kollega) at du forstår hva som skjer. Bekreft at når du logger ut "`teller.onchange`", så får du "`undefined`".

```
» teller.tick();
← 2
» teller.tick();
← 1
» teller.tick();
← 0
» teller.tick();
← undefined
» teller.tick();
← undefined
```

4.b) Endre `tick`-metoden din slik at, *etter* at du har redusert verdien av `counter`-egenskapen, og *før* du returnerer den nye verdien, så kaller du `onchange`-metoden med ett argument: den nye verdien av `counter`-egenskapen.

```
⋮
if (this.onchange) {
  this.onchange(this.counter);
}
⋮
```

```
» teller.tick();
  Ny teller-verdi: 2!
← 2
» teller.tick();
  Ny teller-verdi: 1!
← 1
```

La `onchange`-egenskapen til `teller`-objektet ditt være en funksjon definert med *ett* parameter: nyVerdi. Logg ut den nye verien. Last inn siden på nytt og kall `tick`-metoden til den bare returnerer undefined igjen.

4.c) Endre `tick`-metoden din slik at, *etter* at du har kalt `onchange`-metoden (hvis den finnes), og *før* du returnerer den nye verdien, så kaller du `onzero`-metoden uten argumenter hvis den metoden finnes.

4.d) Opprett en funksjon som sier at telleren har nådd null (en enkel utskrift) og koble den til `teller`-objektet ditt slik at du får melding om dette når du når 0. Hvor mange ganger klarer du å få frem meldingen?

4.e) Du har nå laget et objekt vi kan interagere med, som har en intern tilstand over tid og som kan kalle funksjoner når ulike hendelser inntreffer. Nå skal vi tilrettelegge for litt interaksjon.

Utvid HTML-dokumentet ditt. Legg til en knapp:

```
<button id="myButton">
*
</button>
```

I JavaScript (enten i `oppg4_a.js` eller en ny fil som du inkluderer *etterpå*) definer følgende funksjon:

```
function changeLabel(newLabel) {
  var btn = document.getElementById("myButton");
  btn.innerHTML = newLabel;
}
```

Gjør funksjonskallet `changeLabel("ny verdi");` i konsollen.

4.f) Skriv en funksjon som tar ett argument og setter knappens verdi til å være nøyaktig denne verdien. (Du trenger ikke å beholde `changeLabel`-funksjonen med mindre du vil.) Sett denne funksjonen som tellerens `onchange`-metode. Nå, når du gjør kall til `tick`-metoden (fortsatt i konsollen), vil funksjonen du nå har definert bli kalt. Da endres knappens tekst til å følge tellerens `counter`-verdi.

4.g) Skriv en funksjon som gjør deaktiverer knappen. En slik funksjon vil først finne knappen vha. id-en: `var btn = document.getElementById("myButton");` etterfulgt av å endre et attributt: `btn.disabled = true;`. La denne funksjonen være `onzero`-egenskapen til teller din. Blir knappen deaktivert når telleren når 0?

4.h) La knappen ha attributtet `onclick`. Dette attributtet skal ha verdi som sier at `tick`-metoden på `teller`-objektet skal kalles. Klikk på knappen til den blir deaktivert og følg med på meldingene i konsollen. Får du responsen du forventer?