INFO134 Klientprogrammering

Ukesoppgaver

Uke 4

1 "Klassisk" `for`-løkke

Vi har sett en type `for`-løkke som ligner litt på den vi kjenner fra Python. JavaScript har to typer `for`-løkker. Den ene ligner faktisk mest på `while`-løkken vi kjenner fra før.

1.a) Opprett en tabell `var fam = [p1, p2];` som består av minst to objekter som representerer personer (ta gjerne utgangpunkt i oppgavene for forrige uke).

Skriv ut fornavnene til alle personene ved hjelp av `for`-løkken:

```
for(var personID in fam) {
  var personObj = fam[personID];
  console.log(personObj);
}
```

Til forskjell fra `for`-løkken i Python er det ikke elementet som blir tilordnet itereringsvariabelen, men elementets *indeks*.

Skriv en `for`-løkke som skriver ut alle elementene i listen ("*Array-et*"): `var numbers = [0, 1, 2, 3, 4, 6];`

1.b) For hver liste i variabelen under, skriv ut summen av tallene i den listen:

- `[[1,2], [3, 4], [5, 6]]`. Skal skrive ut tre linjer: `3`, `7`, og `11`.
- `[[1,1], [1,2], [2,3], [3,5], [5, 8], [8, 13]]`

1.c) Oversett oppgavene fra de to forrige oppgavene til en `while`-løkke. (Denne fungerer som i Python, men vi må legge til parenteser og krøllparenteser):

$$\texttt{while}(\overbrace{\texttt{boolean\_expression}}^{\text{iterasjonsvilkår}})\ \{$$
$$\quad \texttt{// kropp}$$
$$\}$$

(Hint: undersøk `length`-egenskapen til listeobjektet når du skal utforme iterasjonsvilkåret.)

1.d) En typisk måte å løse forrige oppgave på er å (1) opprette iterasjonsvariabelen med passende initiell verdi (ofte 0), (2) ha et iterasjonsvilkår sier at vi fortsatt peker på et element i listen, og (3) passer på å oppdatere iterasjonsvariabelen *på slutten* av hver iterasjon.

"Klassisk" `for`-løkke i JavaScript ligner veldig på den vi har i store, tradisjonelle språkene som C, C++, Java, osv.

$$\texttt{for}(\text{initiering}\ ;\ \text{vilkår}\ ;\ \text{oppdatering})\ \{$$
$$\quad \texttt{// kropp}$$
$$\}$$

Gjenta forrige oppgave, men med en klassisk `for`-løkke.

2 Argumenter

2.a) Inspiser `inc`-funksjonen i demonstrasjonen:

`http://wildboy.uib.no/~tpe056/info134/uke4/argumenter.html`

Definer en funksjon med to parametre (`navn` og `hilsen`, eller lignende) som returnerer en tekstverdi. Dersom funksjonen kalles uten argumenter skal funksjonen returnere `Hello, World!`. Dersom første parameter har verdi skal det erstatte "World" i resultatet. Dersom andre parameter også har verdi (altså dersom funksjonen blir kalt med to argumenter) skal det erstatte "Hello" i resultatet.

2.b) Skriv en funskjon, `strangeEven`, som tar opp til tre argumenter.

- Første argument skal være en liste av tall. Dersom bare første argument er angitt skal funskjonen returnere en liste av partallene i første argument.
- Dersom andre argument er gitt skal det være et tall n. Skal funksjonen din bare returnere de n første partallene.
- Dersom *tredje* argument er gitt skal det være en funksjon. Nå skal ikke funksjonen returnere denne funksjonen anvendt på resultatlisten du ellers ville ha fått.

```
» strangeEven([1,2,3,4,5,6]);
← ▸ Array(3) [ 2, 4, 6 ]
» strangeEven([1,2,3,4,5,6], 2);
← ▸ Array [ 2, 4 ]
» strangeEven([1,2,3,4,5,6], 100);
← ▸ Array(3) [ 2, 4, 6 ]
» strangeEven([1,2,3,4,5,6], 2, function(n) {return n-1;} );
← ▸ Array [ 1, 3 ]
```

2.c) Inspiser `greaterOrEqual`-funksjonen i demonstrasjonen over. Definer en funksjon som er definert uten noen parametre (`function logAll() { ... }`) som logger alle argumentene som gis i funksjonskallet.

```
» logAll(1, document, "test");
  Argument 0 :  1
  Argument 1 :  ▸ HTMLDocument file://
  Argument 2 :  test
```

2.d) Definer en funksjon med ett parameter, `bilBeskr`. Vi antar at vi får et objekt som argument. Funksjonen skal returnere en tekstverdi som beskriver en bil.

- dersom `bilBeskr` har egenskapen `farge`, skal bilens farge være med i beskrivelsen.
- dersom `bilBeskr` har egenskapen `eier`, skal dette med i beskrivelsen.
- dersom `bilBeskr` har egenskapen `regNr`, skal dette være med i beskrivelsen.

```
» beskriv({});
← "Bilen finnes."
» beskriv({eier: "Per"});
← "Bilen tilhører Per."
» beskriv({eier: "Per", farge: "blå"});
← "Den blå bilen tilhører Per."
» beskriv({eier: "Per", farge: "blå", regNr: "ST12345"});
← "Den blå bilen har registreringsnummer ST12345 og eies av Per."
» beskriv({eier: "Per", regNr: "ST12345"});
← "Bilen har registreringsnummer ST12345 og eies av Per."
» beskriv({regNr: "ST12345"});
← "Bilen har registreringsnummer ST12345."
```

3 Funksjonsobjekter

3.a) Definer en funksjon (uten parametre) som returnerer 1 første gang den kalles (uten argumenter). Funksjonen skal ha en egenskap som holder telling på hva forrige returnerte tall var og hver gang den kalles, returnerer den neste oddetall.

3.b) Definer en funksjon som alltid returnerer neste heltall (som forrige, men både partall og oddetall). Dersom funksjonen kalles med et argument, skal du sette telleren til det tallet og returnere det.

3.c) Definer en funksjon og gi den fire egenskaper:

`counter` som sier hvilket tall som er det neste som skal skrives us.

`step` som sier hvor mye `counter` skal økes med hver gang.

`limit` som sier hvor høyt vi maksimalt skal øke `counter` til.

`loop` som enten er heltallet vi skal sette `counter` til når vi når grensen, eller `false` dersom vi skal returnere `undefined` *ad infinitum* etter at vi når grensen.

3.d) Definer en funksjon med fire parametre. Funksjonen din skal returnere et nytt objekt som beskrevet i forrige oppgave. Bare `limit` er påkrevet, de andre parameterne skal ha følgende standardverdier: `counter`: 0, `step`: , `loop`: `counter`. Rekkefølge av parametre: `limit`, `counter`, `limit`, `loop`.

```
» c = makeCounter(3);
← ▸ function counterFunction()
» c();
← 0
» c();
← 1
» c();
← 2
» c();
← 0
```

```
» c = makeCounter(10, 1, 4, false);
← ▸ function counterFunction()
» c();
← 1
» c();
← 5
» c();
← 9
» c();
← undefined
```