Opgave 10.1

Lav en constructor function Animal(name,age).

Check, at du kan lave objekter/instanser vha denne.

Tilknyt en function canRun (med en simpel console.log) til denne constructor function (ved at sætte animal.prototype.canRun)

Check, at man kan kalde canRun på de oprettede objekter.

Lav en ny construktor human(name, age, money), med en function canEarn (simpel console.log).

Check igen. Lav to objekter human1 og human2. Check også, at canRun IKKE kan kaldes på human-objekter og vice versa.

Tilret, så human kalder animal ctor i starten af sin egen ctor ved at udføre animal.call(this, name, age);
Dette er I STEDET FOR linierne

```
this.name = name ;

this.age = age ;
```

i human-ctor.

Verificer, at human1 stadig virker.

Link humans prototype til Animals prototype. Dette gøres ved at sætte
`Human.prototype.__proto__ = Animal.prototype`
`Check, at`

`Overskriv canRun på human2-objektet og check, at den gamle definition virker på de andre objekter, men den nye virker på human2`

Opgave 10.2

Tag udgangspunkt i *specialisering.js* og tilføj *equals()* metoder samt en *compare()* metode.

Metoden *equals(p)* på *Person* skal sikre, at *p* har *Person* som *constructor* og har samme *navn* som personen.

Metoden *equals(s)* på *Studerende* skal sikre, at *s* har *Studerende* som *constructor* og har samme *navn* og *id* som den studerende.

Den *static compare(p1, p2)* skal sammenligne *p1* og *p2* baseret på *navn*.

Lav dernæst et array med nogle *personer* og *studerende* og sorter dem på navn.

Indsæt desuden et par *katte* (se *polymorfi.js*) og sorter på ny.

Opgave 10.3

Lav en *class StringStack* med tilhørende *push()* og *pop()* metoder.

Opgave 10.4

Implementer en dobbeltrettet associering mellem *Person* og en *class Gruppe*, som i PRO1.