OBJECTS & PROTOTYPES

Inheritance & shared properties

REPETITION

SCOPE & CONTEXT

Scope kan vara **Global** eller **Function**

Context är vilket objekt som vi refererar till

```
function newScope(){
  //this is now a new scope that
  // we can declare variables in
  if(true){
    // no new scope
  }
}
```

```
function newScope(){
  //this is now a new scope that we can
  //declare variables in
  if(true){
    let i = 0; //new scope
  }
}
```

```
const arrow = () => {
  return "Arrow Function Expression";
}
```

```
const arrow = () => "Arrow"
```

paranteser runt parametrarna om det saknas parametrar eller har flera parametrar

---

return och brackets kan ibland skippas

LEXICAL SCOPE

`=>` binder till lexical scope

Inte bra att använda: som objektmetod t.ex.

Bra att använda: Anonyma funktioner

HAR DU PROBLEM MED EN FUNKTION SOM ANVÄNDER THIS, LOGGA THIS

Börja använda `const` och `let` där det behövs.

OBJECTS

Property: value

OBJEKTETS UPPBYGGNAD

```
let country = {
  country: "Brunei",
  capital: "Bandar Seri Begawan",
  population: 417200
}
```

PROPERTY ACCESS

...the `.` *operator requires an* `Identifier` *compatible property name after it, whereas the* `["..."]` *syntax can take basically any* UTF-8/unicode compatible string *as the name for the property.*

Ref: YDKJ - Contents

```
var obj = {
  bad-name! : "Really Bad Name",
  goodName : "Really Good Name"
};
```

```
bj["bad-name!"]
```

```
bj.goodName
```

Ref: YDKJ - Contents

OBJECT PROPERTIES

Alla objekt beter sig likadant oberoende av innehåll.

Vi kan iterera igenom objektet (loopa)

Vi kan konvertera objektet till JSON t.ex.

Vi kan **INTE** använda t.ex. map

Varför?

SHARED PROPERTIES

Varje funktion/objekt/variabel har inneboende egenskaper som är delade.

Alla strängar kan man använda `.substr()` på

Alla arrayer kan man använda `.sort()` på

En `HTMLCollection` kan man inte använda `.sort()` på.

MOTHER OBJECT

- Alla objekt ärver av Object
- Alla arrayer ärver av Array
- Alla nummer ärver av Number
- Alla strängar ärver av String
- Allt ärver från Object

```
let string = "Hello";
```

```
var string = new String("Hello");
```

Använd alltid den första

Ref: SO - new String();

```
let string = "I am a string"
```

The primitive value "I am a string" is ***not an object*** *. To perform operations on it, such as checking its length, a* String object *is required.*

Ref: YDKJ - Types

Luckily, the language automatically coerces a "string" primitive to a String object when necessary, which means you almost never need to explicitly create the Object form.

Ref: YDKJ - Types

`.defineProperty()`

```
Object.defineProperty(obj, 'property',
            {options});
```

3 värden sätts implicit för varje property

---

writable : går värdet att skriva över

---

enumerable : går det att loopa igenom värdet

---

configurable : om den går att ta bort

```
var obj = { prop: "Hello!"};
```

```
Object.defineProperty( obj, 'prop', {
  value: "Hello!",
  enumerable: false,
  writable: false,
  configurable: false
});
```

Ref: MDN - Object.defineProperty()

PROTOTYPES

.PROTOTYPE PROPERTY

Alla Objekt har en **.prototype** -egenskap

I den finns alla funktioner och värden som är kopplade till varje typ av objekt.

Vi kan använda **.sort()** på varje array eftersom varje array har funktionen **.sort()** i egenskapen .prototype

---

```
Array.prototype.sort()
```

PROTOTYPAL INHERITANCE

if a property or a method is not found in the object itself, then there is an attempt to find this property/method in the prototype chain.

Ref: Js - The Core

PROTOTYPAL INHERITANCE

När ett objekt inte hittar en egenskap letar objektet vidare i prototypkedja tills den når **null**

Liknande som med scope. Hittas inte variabeln i **function scope** letar funktionen vidare i **global scope**

```
var a = {}
```

a --> Object --> null

SKAPA OBJEKT AV ANDRA OBJEKT

```
var a = {};
```

```
var b = a;
```

Båda ärver av **Object.prototype**

Refererar till samma objekt

Detta är dock inte direkt arv

OBJECT.PROTOTYPE

- Object.prototype.constructor()
- Object.prototype.hasOwnProperty()
- Object.prototype.isPrototypeOf()
- Object.prototype.toString()
- Buncha other stuff

OBJECT.CREATE()

```
var a = { x : 5 }
```

Vi skapar ett objekt med egenskapen x med värdet 5

```
var b = Object.create(a)
```

Vi skapar sedan ett objekt baserat på ett annat objekt.

Ref: YDKJ - Prototypes

CONSTRUCTOR FUNCTIONS

```
function Cat(name, type){
  this.name = name;
  this.type = type;
}
```

Function som skapar objekt. Parametrarna sparas i den nya kontexten

---

En vanlig function. Om vi skulle kalla på funktionen som vanligt skulle this vara **window**

```
var a = new Cat("Missy", "Burma");

{
  name : "Missy",
  type: "Burma"
}
```

new måste användas för att ett nytt objekt ska skapas utifrån funktionen och att värderna ska bindas till kontexten.

In other words, in JavaScript, it's most appropriate to say that a "constructor" is any function called with the new keyword in front of it.

Ref: YDKJ - Constructor or call

CAT.PROTOTYPE.MJAU()

```
Cat.prototype.mjau = function(){
  console.log("Mjau I'm " + this.name);
}
```

Alla katter som skapas med `new Cat()` kommer att kunna mjaua

NEW OBJ():

- create new Object() obj
- set obj.__proto__ to Obj.prototype
- return obj;
- Object.create(Obj.prototype)

OBJECT.CREATE(OBJ)

- create new Object() obj
- set obj.__proto__ to Test.prototype
- return obj;

...obj.prototype is in general not the obj's `[[Prototype]]`*. The standard does not provide any way to retrieve the [[Prototype]] property from an object.*

Ref: constructors considered mildly confusing

ETT OBJEKTS EGNA VÄRDEN

Allting ärvs automatiskt om värdet går att ärva.

Prototypkedjan letas igenom tills den tar slut.

Det betyder att vi kan få ut värden ur ett objekt som inte tillhör ett objekt

Vare sig vi vill det eller inte.

FOR...IN

```
for(let prop in obj){
  console.log(prop);
}
```

for..in letar även igenom prototypkedjan vilket betyder att vi kan få ett **false positive**

---

for..of är ES6

ÖVNING

Skapa objekt med `Constructor` och `Object.create`

Finns på GitHub, snart.
