JAVASCRIPT

DESIGN PATTERNS

PROTOTYPER

[[PROTOTYPE]]

Varje objekt har en prototyp:
en egenskap som innehåller gemensamma funktioner och variabler för alla objekt som du skapar.

Varje objekt ärver från **Object**

```
var a = {}
```

```
a --> Object --> null
```

Object.create()

```
var a = {};

var a = Object.create(Object.prototype);

var b = Object.create(null);
```

Om du vill ha ett helt tomt objekt

Ref: Object.create();

```
var a = {};

var b = Object.create(a)

b --> a --> Object --> null
```

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

a --> null
```

FUNCTION.[[PROTOTYPE]]

```
function logContent(){}
```

```
logContent --> Function --> Object --> null
```

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

DESIGN PATTERNS

A pattern is a reusable solution *that can be applied to commonly occurring problems in software design - in our case - in writing JavaScript web applications.*

Ref: What is a Design Pattern

VAD ÄR ETT MÖNSTER?

Problem: Jag måste skapa jättemånga olika objekt som följer samma struktur.

```
function Country(name, population){
  this.name = name;
  this.population;
}
```

Lösning: Jag skapar en funktion som skapar objektet åt mig.

Problem: Jag måste gruppera ett antal funktioner som logiskt hört samman.

```
var CountryDatabase = {
  dataAboutCountries: [],
  functionUsingTheData : function(){
  }
}
function functionThatDoesNotUseTheData(){
  //I'm a function that does another thing
}
```

1. Try stuff until it works.

2. Find a "pattern" or "paradigm" after the fact that loosely justifies what you did.

ANVÄNDNINGSOMRÅDEN

När ska jag använda vad?

"Ugh, hittar ingenting och jag har döpt massa grejer till samma sak så jag är osäker på vilken kod jag kallar på."

Du behöver troligen strukturera din kod bättre, t.ex. med beprövade metoder.

Samma som med allt annat: behöver inte följas strikt, de är bara hjälpmedel.

OBEROENDE AV SPRÅK

Oberoende av programmeringsspråk har nästan alla samma sort problem: svårt att lägga upp koden.

Du kommer stöta på samma sorts mönster i andra språk men med olika implementationer: språk som är lika varandra har liknande problem.

Jag tar upp några JavaScript-implementationer som man kan använda sig av.

CONSTRUCTOR PATTERN

```
function Person(name, age, vegan){
  this.name = name;
  this.age = age;
  this.vegan = vegan;
}
```

Vi har ju flera olika sätt att skapa objekt, **Constructor pattern** är ett av dessa vi kan använda.

```
Person.prototype.isVegan = function(){
  return this.vegan;
}
Person.prototype.sayName = function(){
  console.log("I'm " + this.name);
}
```

Det skapas ett **prototype**-objekt som är kopplat till varje nytt objekt.

Ärvda properties

PROTOTYPE PATTERN

```
var prototype = {
  prototypeFunction: function(){
    console.log("Hello from prototype!");
  }
}
var a = Object.create(prototype);
```

Vi skapar ett objekt baserat på ett annat objekt, ärver det första objektets funktioner.

Behöver inte instansiera med `new`

CONSTRUCTOR VS. PROTOTYPE

PATTERN

Använd den som känns mest logisk för dig

Med Object.create måste man skapa en egen constructor

Människor som är `true` till JavaScript brukar förespråka att man inte ska använda `new` t.ex.

Båda utför jobbet, välj den som gör jobbet enkelt för dig.

IMMEDIETLY-INVOKED FUNCTION EXPRESSION

IIFE

Ref: Wiki - IIFE

IIFE

```
(function(){
  console.log("I run and log directly!");
  console.log("I don't have to be called by you!"
  console.log("Praise our robot overlords");
})();
```

IIFE

```
(function(){
  return "I get returned but not saved :(";
})();
```

```
var saved = (function(){
  return "Yay, I now exists in the variabel 'save
})();
```

IIFE

Använd när du behöver skapa closures för att bevara variabler

Använd när du har kod som t.ex. ska kallas på automatisk: init-funktion

Funktionen körs direkt efter att den är skapad

Ligger den däremot inuti en annan funktion körs den inte förän den yttre funktionen kallas på

PARAMETERS

```
(function(param){
  console.log(param + ' So anonymous!');
})("Wow!");
//"Wow! So anonymous!"

(param => {
  console.log(param + ' So anonymous!');
})("Wow!");
//"Wow! So anonymous!"
```

MODULE PATTERN

Problem: All min kod är tillgänglig överallt. Jag vill följa Principle of Least Privilege

```
var obj = {
  importantProp: "importantValue"
};
```

```
obj.importantProp = "";
```

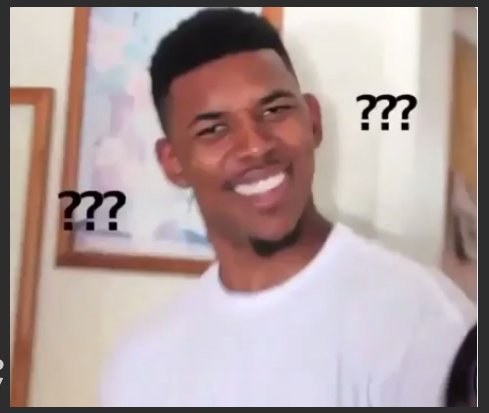
Inte så viktig längre

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

Omständligt och vi kanske vill ändra värdet i framtiden.

Vi vill kunna skriva över värdet, men det ska inte kunna ske av misstag.

MODULE PATTERN




```
var Module = (function(){
  var importantProp = "importantValue";
  return {
    getImportantValue: function(){
      return importantValue;
    }
  }
})();
```

En IIFE som direkt returnerar ett objekt. importantProp finns bara i en closure

REVEALING MODULE PATTERN

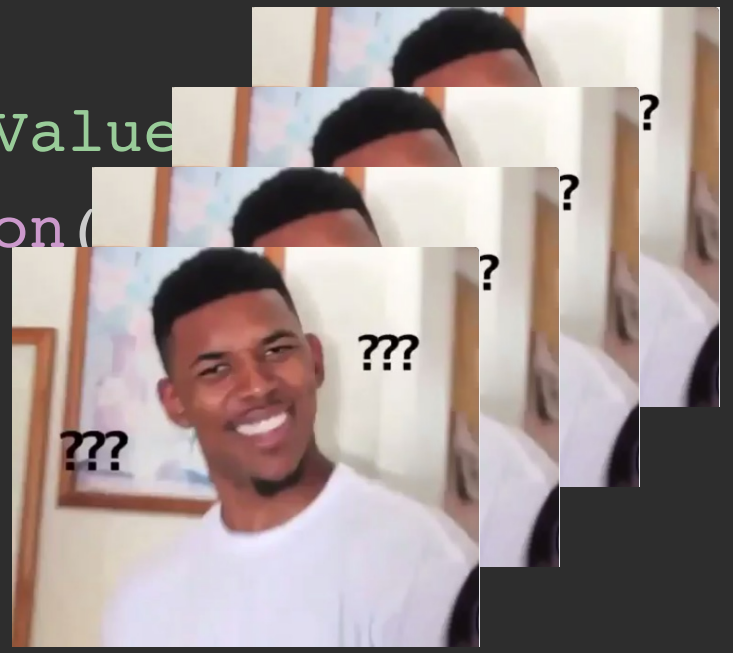
REVEALING MODULE PATTERN

Revealing module pattern är en variant på Module Pattern

Vi får här välja vilka funktioner och variabler som blir revealed : synliga för resten av koden

Privata funktioner kan heta något helt annat än publika funktioner

```
var Module = (function(){
  var importantProp = "importantValue
  var getImportantValue = function(
      return importantValue;
  }
  return {
    getIt: getImportantValue
  }
})();
```



I objektet som returneras så väljer vi vad vi ska returnera och efter vilket namn

NAMESPACES

Namespaces can be considered a logical grouping of units of code under a unique identifier.

```
var namespace = {
  prop: val
}
```

```
namespace.prop === val
```

In JavaScript, they help us avoid collisions *with other objects or variables in the global namespace. They're also extremely useful for helping* organize blocks of functionality *in a code-base so that it can be more easily referenced and used.*

Ref: Namespacing Patterns

Koden vi arbetar med är inte alltid vår egen kod

Det är inte alltid lätt att få en överblick över vad som är deklarerat redan.

När man använder bibliotek eller plugins t.ex.

Namespacing any serious script or application is critical as it's important to safeguard our code from breaking *in the event of another script on the page using the same variable or method names we are.*

Ref: Namespacing Patterns

NAMSPACING PATTERN

Object Literal, Module Pattern och Revealing Module Pattern är olika mönster för att hantera namespaces

Constructor & **Protoypes** är olika mönster för att skapa objekt

Allt använder sig av objekt och prototyper, scope och context.

ÖVNING

Implementera Module Pattern och Revealing Module Pattern