

Funktioner

Finns det redan funktioner?

✓ När ett JavaScript-program startar finns det inbyggda funktioner (och variabler).

✓ `console.log` är en funktion som skriver ut ett värde i konsolfönstret.

```
let urging = '"Use the forks Luke!", Obi-Wan said in the Chinese restaurant.'  
console.log(urging) // Output: "Use the forks Luke!", Obi-Wan said in the Chinese restaurant.
```

✓ `Math.min` är en funktion som returnerar det minsta av flera värden.

```
let min  
min = Math.min(103, 42, 57)  
console.log(min) // Output: 42
```

Vad är funktioner bra för?

- ✓ Alla språk har inbyggda funktioner som utför förutbestämda saker.

```
console.log('Hej hopp!') // skriver ut en sträng i konsolfönstret  
let result = Math.floor(3.14) // avrundar ett flyttal till närmsta lägre heltal
```

- ✓ Det finns dock inte funktioner för alla saker; du måste skriva egna.
- ✓ Att dela upp ett program i mindre delar, funktioner, är viktigt för att...
 - ...strukturera kod.
 - ...få överblick över koden.
 - ...undvika att identisk kod upprepas. Bryt aldrig mot principen DRY (Don't Repeat Yourself!).
 - ...koden blir enklare att testa.

Så anropar du en funktion

- ✓ Då en funktion anropas exekveras den.

```
console.log('Hello, World!') // Skriver ut strängen (och returnerar värdet undefined).
```

```
console.log() // Skriver ut en ny rad (och returnerar värdet undefined).
```

```
let randomNumber = Math.random() // Returnerar ett värde från och med 0 till 1.
```

- ✓ En funktion anropas genom att ange dess namn följt av parenteser innehållande eventuella argument.
 - Argument är värden som skickas till funktionen. Det är funktionen som bestämmer om inget , ett eller flera argument används då funktionen exekveras.
 - Parenteserna efter funktionens namn måste alltid anges vid anrop, oavsett om det finns argument eller inte.
- ✓ Då en funktion exekverats klart returneras alltid ett värde, ett så kallat returvärde.
 - Returvärde är det värde en funktion ger tillbaka. Bara ett värde kan returneras.

Så skapar du en funktion med en funktionsdeklaration

- ✓ En funktionsdeklaration består av det reserverade ordet `function`, följt av:
 - Funktionens namn (som följer samma regler som gäller för variabler).
 - Ett par parenteser runt en kommaseparerad lista med parametrar (vars namn följer samma regler som gäller för variabler), som fungerar som lokala variabler i funktionen. En funktion kan ha inga, en eller flera parametrar.
 - Ett par klammerparenteser innehållande inga, en eller flera satser som utgör funktionens kropp. Satserna exekveras då funktionen anropas.
 - `return`-satsen, om den finns någon, bestämmer värdet funktionen returnerar då den anropas.
 - Saknas `return`-sats returneras värdet `undefined`.

```
function sayHelloTo (name) { // Parametern name tar hand om argumentet som skickas med vid anrop av funktionen.  
  let greeting = 'Hello, ' + name + '!'  
  return greeting  
}
```

```
let whatIsSaid = sayHelloTo('Ellen Nu') // Argumentet 'Ellen Nu' kopieras till parametern name vid anropet av funktionen.  
console.log(whatIsSaid) // OUTPUT: Hello, Ellen Nu!
```

```
whatIsSaid = sayHelloTo('Nisse Hult') // Argumentet 'Nisse Hult' kopieras till parametern name vid anropet av funktionen.  
console.log(whatIsSaid) // OUTPUT: Hello, Nisse Hult!
```

Funktioner kan skapas på flera sätt, men anrop, argument, parametrar och returvärden fungerar på samma sätt oavsett sätt hur en funktion skapas.

Så skapar du en funktion med ett funktionsuttryck

- ✓ Du definierar en funktion med ett funktionsuttryck på ungefär samma sätt som en variabel, värdet är bara en funktion.

- Du behöver inte ange något namn på funktionen.

```
const anotherSayHelloTo = function (name) {  
  let greeting = 'Hello, ' + name + '!'  
  return greeting  
}
```

```
let whatIsSaid = anotherSayHelloTo('Ellen Nu')  
console.log(whatIsSaid) // OUTPUT: Hello, Ellen Nu!
```

- ✓ Ett funktionsuttryck deklarerar en konstant och tilldelar den ett funktionsobjekt.
 - Använd `const` i samband med funktionsuttryck för att undvika att funktionen skrivs över med ett nytt värde.
- ✓ Funktioner definierade med funktionsuttryck kan inte anropas innan de är initierade.
 - (En funktion skapad med en funktionsdeklaration skapas innan koden som innehåller den exekveras.)

Så skapar du en funktion med ”*arrow function*”-syntax

- ✓ Från och med ES6 kan du skapa funktioner med ”*arrow function*”-syntaxen.

```
const yetAnotherSayHelloTo = (name) => {  
  let greeting = 'Hello, ' + name + '!!'  
  return greeting  
}
```

Parenteserna behövs egentligen inte om det bara är en parameter, men ta för vana att alltid skriva dem.

```
let whatIsSaid = yetAnotherSayHelloTo('Ellen Nu')  
console.log(whatIsSaid) // OUTPUT: Hello, Ellen Nu!
```

- ✓ Innehåller funktionens kropp bara en sats kan `return` uteslutas.

```
const yetAnotherSayHelloTo = (name) => 'Hello, ' + name + '!!'  
  
let whatIsSaid = yetAnotherSayHelloTo('Ellen Nu')  
console.log(whatIsSaid) // OUTPUT: Hello, Ellen Nu!
```

- ✓ Saknar funktionen parametrar måste parenteser användas.

```
const getTheMeaningOfLife = () => 42
```

Funktioner och räckvidd – ”*scope*”

- ✓ Variabler, konstanter och parametrar har alla en räckvidd. Räckvidden bestämmer var de går att använda.

```
function getFullName (firstName, lastName) { // Parametrarna är lokala i funktionen.  
  let fullName = firstName + ' ' + lastName // 'fullName' är en lokal variabel.  
  return fullName  
}
```

```
let name = getFullName('Ellen', 'Nu')  
console.log('Personens fullständiga namn är ' + name + '.')  
// OUTPUT: Personens fullständiga namn är Ellen Nu.
```

```
const fName = 'Nisse'  
const lName = 'Hult'  
name = getFullName(fName, lName)  
console.log('Personens fullständiga namn är ' + name + '.')  
// OUTPUT: Personens fullständiga namn är Nisse Hult.
```

- ✓ Parametrarna `firstName` och `lastName` är lokala i funktionen `getFullName`, och går bara att använda i funktionen.
- ✓ Variabeln `fullName` definieras med `let` och är därmed lokal i funktionen `getFullName`.
- ✓ Variabler skapade med `let`, eller konstanter, har det som kallas ”*block scope*”. De går bara att använda i blocket, mellan klammerparenteserna, de befinner sig i. (var ger det som kallas ”*function scope*”.)

Parametrar och standardvärden

- ✓ Då en funktion anropas med färre argument än antalet parametrar sätts parametrars värden till standardvärdet `undefined`, om inget annat angivits.

```
function getFullName (firstName, lastName = 'Doe') {  
  let fullName = firstName + ' ' + lastName  
  return fullName  
}
```

Första parametern saknar angivet standardvärde och får därför standardvärdet `undefined`. Andra parametern har tilldelats standardvärdet `'Doe'`.

```
let name = getFullName('Ellen', 'Nu')  
console.log('Personens fullständiga namn är ' + name + '.')  
// OUTPUT: Personens fullständiga namn är Ellen Nu.
```

Två argument anges och inga standardvärden kommer att användas.

```
name = getFullName('Nisse')  
console.log('Personens fullständiga namn är ' + name + '.')  
// OUTPUT: Personens fullständiga namn är Nisse Doe.
```

Ett argument anges och standardvärdet `Doe` kommer att användas.

```
name = getFullName()  
console.log('Personens fullständiga namn är ' + name + '.')  
// OUTPUT: Personens fullständiga namn är undefined Doe.
```

Inga argument anges varför standardvärdena `undefined` och `Doe` kommer att användas.

