

# **Comp 125 - Visual Information Processing**

---

Spring Semester 2019 - Week 7 - Monday

Dr Nick Hayward

# JavaScript - functions - basic usage - part 4

---

## **parameters and arguments**

- custom functions may also be modified by defining accepted **parameters**
  - *parameter values may be used in the executed logic*
- parameters allow a developer to pass values into the function
  - *may be used to modify the logic and executed code*
- parameters are always defined between a function's parentheses
- as we call the function, we pass the required values as **arguments**
  - *also specified between the parentheses for the function call*

# JavaScript - functions - basic usage - part 5

## using parameters and arguments - example

- structure for a function with parameter

```
function (parameter) {  
    // test output of parameter  
    console.log("function parameter = " + parameter);  
}
```

- example usage might be as follows

```
function sayHello(name) {  
    // output greeting to person  
    console.log('Hello' + name + ', how are you?');  
}
```

- then call this function
  - passing an argument for the required function parameter

```
sayHello('Amelia');
```

## JS Functions - parameters and arguments - example

---

add a custom function with a parameter, and call function with passed argument...

The screenshot shows a browser's developer tools console window. At the top, there are tabs for 'top' and 'Filter', along with settings for 'Default levels' and 'Group similar'. The main area of the console displays the following JavaScript code and its execution:

```
> // define function
  function sayHello(name) {
    console.log('Hello ' + name + ', how are you?');
  }

// call function by name
sayHello('Amelia');
Hello Amelia, how are you? VM1382:3
< undefined
> |
```

JS - function call 3

# JavaScript - functions - basic usage - part 6

---

## ***using parameters and arguments - multiple***

- functions may also specify multiple parameters
- function calls may pass multiple arguments

```
function nameGenerator(first, last) {  
    ...  
}
```

- each parameter is separated by a comma
- function body may use either or both parameter

# JavaScript - functions - basic usage - part 7

---

## using parameters and arguments - multiple

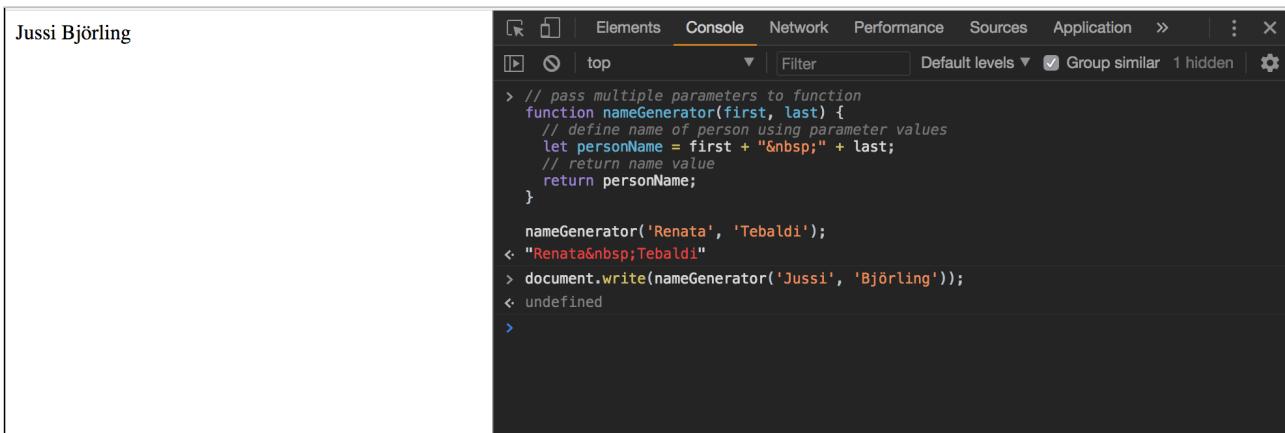
```
// pass multiple parameters to function
function nameGenerator(first, last) {
  // define name of person using parameter values
  let personName = first + " " + last;
  // return name value
  return personName;
}

nameGenerator('Renata', 'Tebaldi');
```

# JS Functions - parameters and arguments - example

---

custom function with multiple parameters...



The screenshot shows the 'Console' tab of a browser's developer tools. The user has defined a function named 'nameGenerator' that takes two parameters, 'first' and 'last'. It concatenates them with a space and returns the result. The user then calls this function with the arguments 'Renata' and 'Tebaldi', and logs the result to the console. Finally, the user uses 'document.write' to output the name to the page.

```
// pass multiple parameters to function
function nameGenerator(first, last) {
    // define name of person using parameter values
    let personName = first + " " + last;
    // return name value
    return personName;
}

nameGenerator('Renata', 'Tebaldi');
<- "Renata Tebaldi"
> document.write(nameGenerator('Jussi', 'Björling'));
<- undefined
>
```

JS - function with multiple parameters

## JS Functions - functions as values

---

- return value from a function may be used within a block of code
- function calls may be used as values
- enables dynamic values for variables &c.

```
// generate random number using min and max values
function randomNumber(min, max) {
    // get random number
    let number = Math.floor(Math.random() * (max - min + 1)) + min;
    // return random number
    return number;
}
// call random number
var num = randomNumber(5, 17);
```

# JS Functions - functions as values - example

use function as value...

The screenshot shows a browser's developer tools console window. The title bar includes 'Elements', 'Console' (which is selected), 'Sources', and 'Network'. Below the title bar, there are tabs for 'top' and 'Filter' with a dropdown for 'Default levels' and a checked checkbox for 'Group similar'. The main area of the console displays the following JavaScript code:

```
// generate random number using min and max values
function randomNumber(min, max) {
    // get random number
    let number = Math.floor(Math.random() * (max - min + 1)) +
    min;
    // return random number
    return number;
}
// call random number
var num = randomNumber(5, 17);
// output random number
document.write('random number = ' + num);
```

At the top left of the code area, the text 'random number = 9' is displayed, indicating the result of the function execution.

JS - function as value

# JS Functions - functions as argument

---

- pass function as argument
  - get *return value from passed function*
- e.g. call another function with random number as argument
  - *dynamic value will always be passed to function as argument*

```
// count down from passed value
function countDown(count) {
    // use count in for loop
    for (i = count; i <= count; i--) {
        // log value of each iteration of loop
        console.log('<br>count = ' + i);
        // check end of count
        if (i === 0) {
            return true;
        }
    }
}
// use random number as argument to countDown
countDown(randomNumber(3, 13));
```

# JS Functions - functions as values - example

use function as argument...

```
counter = 10

count = 10
count = 9
count = 8
count = 7
count = 6
count = 5
count = 4
count = 3
count = 2
count = 1
count = 0

counter = 3

count = 3
count = 2
count = 1
count = 0

// generate random number using min and max values
function randomNumber(min, max) {
    // get random number
    let number = Math.floor(Math.random() * (max - min + 1) +
min);
    // return random number
    return number;
}

// count down from passed value
function countDown(count) {
    // output count value
    document.write('<h3>counter = ' + count + '</h3>');
    // use count in for loop
    for (i = count; i <= count; i--) {
        document.write('<br>count = ' + i);
        // check end of count
        if (i === 0) {
            return true;
        }
    }
}

// use random number as argument to countDown
countDown(randomNumber(3, 13));
< true
> // use random number as argument to countDown
countDown(randomNumber(2, 8));
< true
> |
```

JS - function as value

# HTML - markup for headings - part I

---

- HTML is flexible in markup usage
  - due to *presentational versus structural considerations*
- headings might be perceived as purely presentational, e.g.

```
<span class="heading">Chapter 1</span>
```

- issues with presentational markup, e.g.
  - visual browsers with CSS will render as expected
  - no CSS, and browsers will render as normal text
  - non-visual browsers = normal text and no heading
  - accessibility issues...
- search engines, ranking, spiders...
  - will not process this markup as a heading
  - no semantic meaning...
  - recorded as normal text
- CSS styles can be unique
  - but restricted to *class* usage with heading

## HTML - markup for headings - part 2

---

- many different ways to markup content with HTML, e.g.

```
<p><b>Chapter 1</b></p>
```

- issues still exist with variant markup options, e.g.
  - *visual browsers will render text in bold & same size as default*
  - *unique styling is problematic...*
  - *search engines, ranking, spiders...*
    - will not process this markup as a heading
    - no semantic meaning...
    - recorded as normal text

# HTML - markup for headings - part 3

---

- use markup correctly with structure and meaning, e.g.

```
<h3>Chapter 1</h3>
```

- benefits of this markup, e.g.
  - *conveys meaning to contained text*
  - *visual and non-visual browsers treat heading correctly*
    - regardless of any associated styles...
  - *easy to add unique styles with CSS*
  - *search engines &c. will interpret this markup correctly*
    - extract keywords, semantics, structure...