Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 6 - Friday

Dr Nick Hayward

Fun exercise - using objects

- create an object or objects with information about an archive
  - *include name and location of the archive*
- use a combination of arrays and objects to store information about books in the archive - minimum five books
  - *include author's name, book title, date of publication, number of pages...*
- output to the document all of the names of the books in the archive
  - *output to the document all information for at least one book in the archive*

Output answers to the document with link breaks between results.

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 4

- use conditional statement to check letter
  - *check against `gameWord` - should return `true` boolean*
  - *check against `answers` - should return `false` boolean*

```
// check letter against game word & not in answers - check for duplicate letter g
if (gameWord.includes(letter) === true && answers.includes(letter) === false) {
    ...
} else {
    ...
}
```

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 5

- then use `for` loop through `gameWord`
  - *check guess letter against each letter in gameWord*
  - *use loop index `i` to check each value in gameWord*

```
// loop through gameWord
for (i = 0; i < gameWord.length; i++) {
    // check letter against each value in gameWord
  if (gameWord[i] === letter) {
        // add letter to answers array at matching index position
    answers[i] = letter;
    }
}
```

- add guess letter to answers array using loop index `i`

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 6

- also need to keep a record of wrong letter guesses
- use `lettersToGuess` variable
- value is initially set to length of game word

```
// set value for letters to guess from random word
var lettersToGuess = gameWord.length;
```

- then decrement in loop for letter check in `gameWord`

```
lettersToGuess--;
```

HTML & JavaScript - create a game - check guess letter

check letter against game word - part 7

- use `lettersToGuess` to check for end of game
    - *player wins if value reaches 0*

```
// check if gameWord has been guessed correctly
if (lettersToGuess === 0) {
    console.log('game over...player won');
    document.getElementById('guessLetter').innerHTML = 'GAME OVER: word guessed c
    // exit game and reset...need to add
}
```

HTML & JavaScript - create a game - verbose working example

conditional statement and `for` loop

```
// check letter against game word & not in answers - check for duplicate letter g
if (gameWord.includes(letter) === true && answers.includes(letter) === false) {
  console.log('letter has been found...' + gameWord.includes(letter));
  // loop through gameWord
  for (i = 0; i < gameWord.length; i++) {
    // check letter against each value in gameWord
    if (gameWord[i] === letter) {
      console.log('letter = index ' + i);
      // add letter to answers array at matching index position
      answers[i] = letter;
      // decrement remaining letters to guess to win game...
      lettersToGuess--;
      console.log('letters left to find = ' + lettersToGuess);
      // update game progress to player
      var lettersOutput = answers.join(" "); // create string from answers array
      document.getElementById('wordStatus').innerHTML = 'guess word: ' + lettersO
    }
  }
  // check if gameWord has been guessed correctly
  if (lettersToGuess === 0) {
    console.log('game over...player won');
    document.getElementById('guessLetter').innerHTML = 'GAME OVER: word guessed c
    // exit game and reset...need to add
  }
} else {
  console.log('letter not found...');
  document.getElementById('guessLetter').innerHTML = 'letter not found - please t
  // draw output to hangman...need to add
}
```

- Hangman Game - v0.3

HTML & JavaScript - create a game - restart game

reset game and load new game word

- need to reset the game after **GAME OVER**
  - *player wins or loses...*
- game requires reloading, resetting of variables, data structures...
  - *might use simple browser refresh*
  - *better option is to dynamically reset game logic*
- need to abstract code to **functions**...

HTML & JavaScript - create a game

work left to complete

- code is **too** verbose
- code needs **abstraction**
- need to introduce **functions** for better code structure and reuse
- **reset** option necessary for **GAME OVER**
- hangman figure needs to be drawn to HTML document
- small updates to usability
  - *clear letter in input field after `guess` button pressed*
  - *add event listener for **return** key press in input field*
  - *add autofocus to input field*

HTML & JavaScript - create a game - quick updates

update usability on input field

- update event listener for mouse click on `guess` button
- reset value for input field after click event
    - *use empty string to clear input field*
    - *placeholder text will then be shown in input field*

```
// reset input field
document.getElementById('guess').value = "";
```

- reset focus on input field after click event

```
// reset focus on input field
document.getElementById('guess').focus();
```

JavaScript - functions - intro

- game code needs **LOTS** of abstraction and refactoring
- functions are a great way to help such abstraction and reuse
- a **function** is a common and useful option for grouping code
  - *organise for reuse within an application*
- reuse of functions also helps provide better abstraction of logic
- group and store functionality in JS functions
  - *use repeatedly by calling the same function*
- functions also help us organise our code and application logic
  - *providing better structure and design to our code*
- functions help us test our application code more easily
  - *creating manageable chunks of code and logic*
- we may also define accepted parameters for a functio
  - *enabling customisation and broader usage of contained code and logic*
- return values for a given function may be customised
  - *relative to passed arguments as we call a function*

JavaScript - functions - basic structure

- basic structure for function syntax

```
function () {
    ...code to excute...
}
```

- we can extend this syntax
  - *add a **name** for the function*
  - *define accepted **parameter** (or parameters)*
  - *use and return code from a function...*

JavaScript - functions - basic usage - part 1

define function with name and parameter

- add a custom name for a function
  - *this function will log a string to the console...*

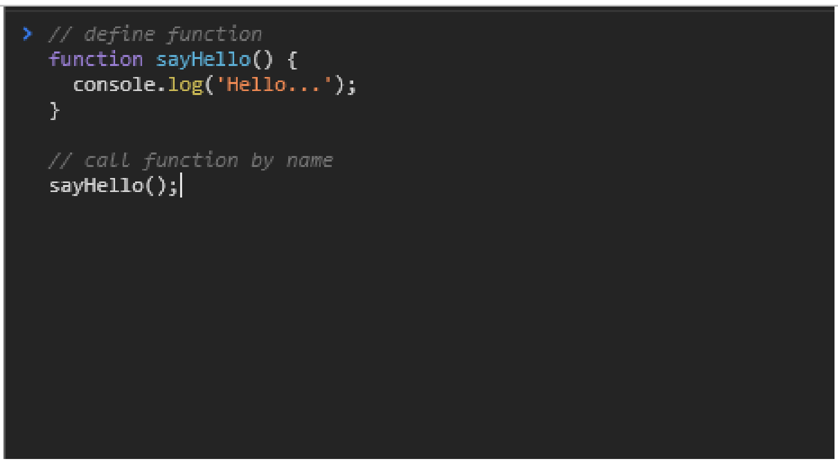
```
function sayHello () {
    console.log('Hello...');
}
```

- execute this code by calling the function's name
  - *add parentheses to denote name as function*

```
sayHello();
```

JS Functions - name and call

add a custom function name and call...



```
> // define function
  function sayHello() {
    console.log('Hello...');
  }

  // call function by name
  sayHello();
```

JS - function call 1

JavaScript - functions - basic usage - part 2

define function as value of variable

- also assign a function as the value of a named variable

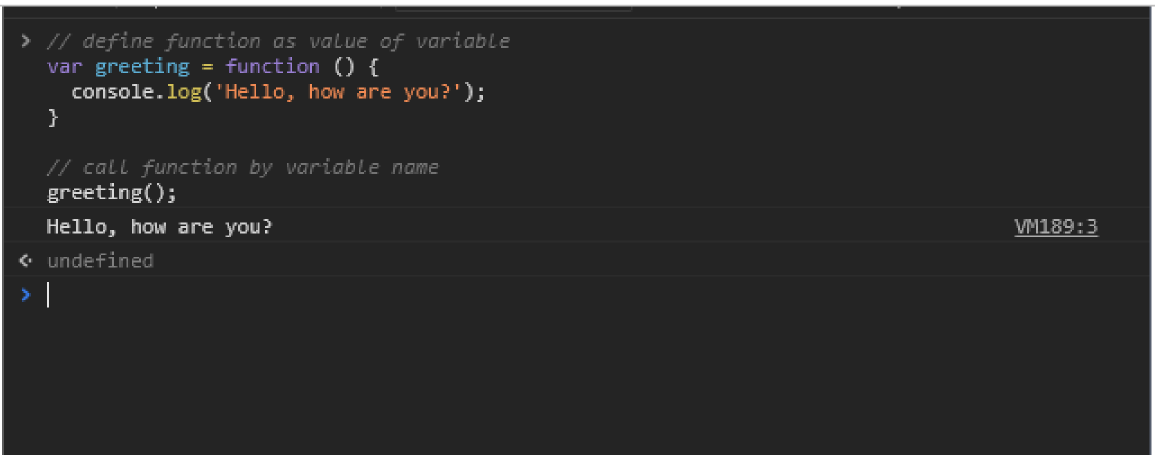
```
var greeting = function () {
    console.log('Hello, how are you?');
};
```

- then call this function using the same pattern

```
greeting();
```

JS Functions - name and call - example 2

add a custom function name and call as value of variable...



```
> // define function as value of variable
  var greeting = function () {
    console.log('Hello, how are you?');
  }

  // call function by variable name
  greeting();
  Hello, how are you?                                                   VM189:3
<· undefined
> |
```

JS - function call 2

JavaScript - functions - basic usage - part 3

***`return`* value**

- previous examples included a `return` value of `undefined`
- `return` value is value that a function will actually output
  - *for reuse elsewhere in the application*
- `console.log()` returns its own value
  - *not value for custom function*
- return value will always be `undefined`
  - ***unless** we specify a `return` value for the function*

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

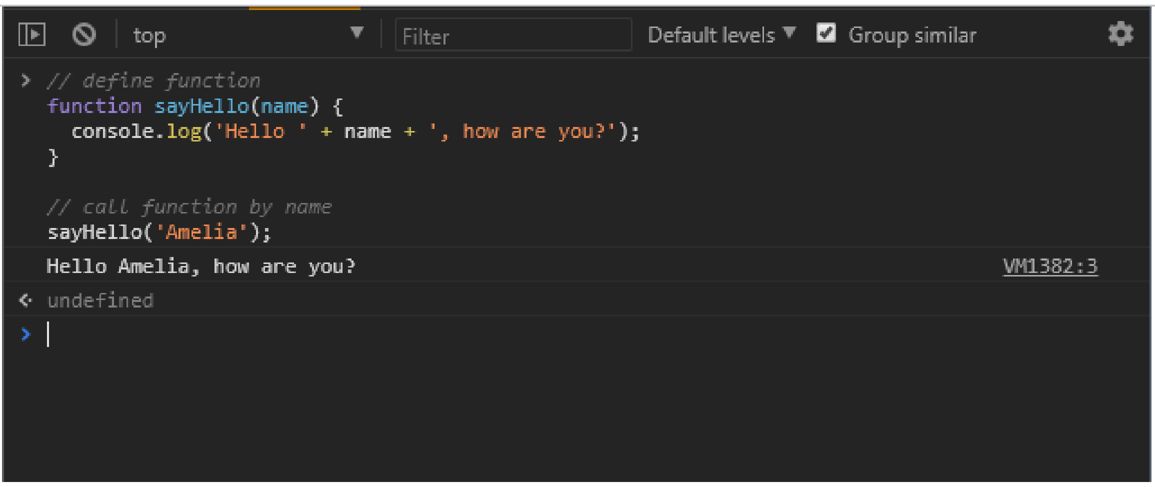
```
function sayHello(name) {
    // output greeting to person
    console.log('Hello' + name + ', how are you?');
}
```

- 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...



```
> // 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

References

- W3Schools
  - *JS - conditionals*
  - *JS - For loop*
  - *JS - functions*