

Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 3 - Monday

Dr Nick Hayward

JS Basics - data types - extras

- two more data types to consider, e.g. undefined and null
 - *undefined*
 - a variable declared or updated without a value is **undefined**
 - its data type will also be **undefined**
 - e.g.

```
// variable declared without value and data type
var greeting;
// update variable to empty - specify value and type as `undefined`
greeting = undefined;
```

- null
 - sets the value of a variable to **nothing**
 - data type will be set to **object** (this is known bug in JavaScript)
 - e.g.

```
// declare variable with value set to nothing - type will be `object`
var greeting = null;
```

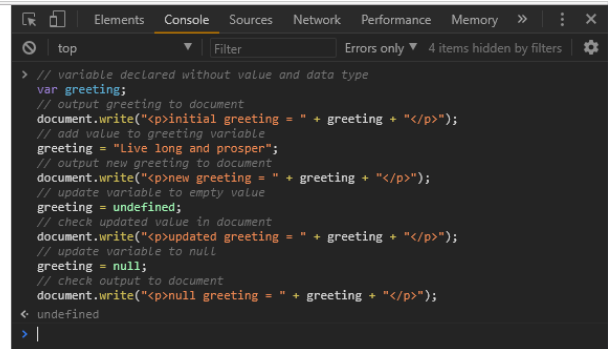
JS Basics - data types - undefined vs null

- there is a difference between `undefined` and `null`
- difference is the **data type**
- `undefined` data type = **undefined**
- `null` data type = **object**
- both values will return nothing - i.e. they will be **empty**
- data types will return different results

JS Basics - data types - extras

Declare variables with undefined and null...

```
initial greeting = undefined  
new greeting = Live long and prosper  
updated greeting = undefined  
null greeting = null
```



The screenshot shows a web browser's developer console with the 'Console' tab selected. The code being executed is as follows:

```
> // variable declared without value and data type  
var greeting;  
// output greeting to document  
document.write("<p>initial greeting = " + greeting + "</p>");  
// add value to greeting variable  
greeting = "Live long and prosper";  
// output new greeting to document  
document.write("<p>new greeting = " + greeting + "</p>");  
// update variable to empty value  
greeting = undefined;  
// check updated value in document  
document.write("<p>updated greeting = " + greeting + "</p>");  
// update variable to null  
greeting = null;  
// check output to document  
document.write("<p>null greeting = " + greeting + "</p>");
```

The console output shows the following sequence of HTML elements being written to the document:

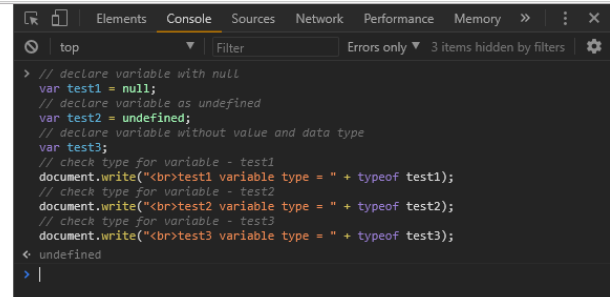
```
< p>initial greeting = undefined  
< p>new greeting = Live long and prosper  
< p>updated greeting = undefined  
< p>null greeting = null
```

JavaScript - undefined and null

JS Basics - data types - check typeof

Use typeof operator to check data type...

```
test1 variable type = object  
test2 variable type = undefined  
test3 variable type = undefined
```



```
> // declare variable with null  
var test1 = null;  
// declare variable as undefined  
var test2 = undefined;  
// declare variable without value and data type  
var test3;  
// check type for variable - test1  
document.write("<br>test1 variable type = " + typeof test1);  
// check type for variable - test2  
document.write("<br>test2 variable type = " + typeof test2);  
// check type for variable - test3  
document.write("<br>test3 variable type = " + typeof test3);  
undefined  
> |
```

JavaScript - check data type

JS Basics - naming variables

- we need to be careful as we enter variable names
 - *misspell a variable name and JavaScript will return an error*
 - known as a `ReferenceError`
- variable names may not contain spaces
 - *a basic use of multiple words, e.g.*

```
var travelbook = "Hannibal's Footsteps";  
var noofwords = 1997;
```

- difficult to read variable name with this style
 - ***camel case*** is preferred style for multiple word variable names, e.g.
- each word's first character is capitalised
 - *convention for variable names is lowercase for first character*
 - using ***camelCase*** we can write our variables as follows,

```
var travelBook = "Hannibal's Footsteps";  
var noOfWords = 1997;
```

Fun exercise - using variables and operators

- calculate the **number of seconds in an hour**
- using the **number of seconds in an hour**, calculate the **number of seconds in a day**
- using **number of seconds in a day**, calculate the **number of seconds in a year**
- using **number of seconds in a year**, calculate the **number of seconds in your current age** in years, e.g. 22 years

Output each answer to the document with a line break between each result.

JS Data Structures - intro

- store data values as individual values in a single variable
 - *strings, numbers...*
 - *useful for storing a word, phrase...*
- we also need to be able to store large amounts of data
 - *e.g. multiple values in a single variable*
- large amounts of data will need to be organised, e.g.
 - *a numerical index of values*
 - *a key/value pair to reference and search values*
- large amounts of data can be stored in **data structures**
- data structures in JavaScript
 - *indexed collections - **arrays**...*
 - *keyed collections - **maps, sets**...*

Further details,

- MDN - JavaScript data types and data structure

JS Data Structures - arrays - intro

- an array allows us to store multiple values in a single variable
 - *includes associated index, and various object properties such as length*
- arrays are one of the most common data types and structures in programming
- using an array, we may now handle various collections of items
- e.g. names in a sports team in an array instead of separate variables
- the size of an array is also dynamic, e.g.
 - *add a new player's name to the array*
 - *remove an existing name from the array*
- arrays are **objects** in JavaScript
 - *provides access to functions (methods) to work with arrays*
 - *arrays include their own properties as well, e.g. `length`*

Further details,

- W3Schools - Arrays
 - *MDN - Array*

JS Data Structures - arrays - creating an array

- create an array in JavaScript using two options,
 - *using the built-in Array constructor*
 - *using array literals []*

```
// using array literals to create new array
var players = ["Amelia", "Emma", "Daisy", "Yvaine"];
// using Array constructor to create new array
var places = new Array("Paris", "Nice", "Marseille");
```

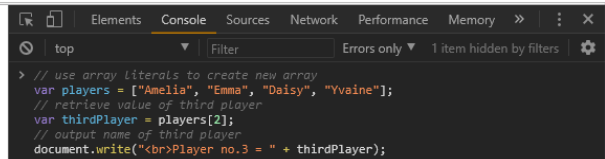
- array literals are more common option for creating new array
 - *Array constructor useful for extending and customising array properties &c.*
 - *offers advanced options for customisation...*

JS Data Structures - arrays - access

- use **index** of an array to retrieve stored values, e.g.

```
players[0];  
"Amelia"  
places[1];  
"Nice"
```

Player no.3 = Daisy



```
> // use array literals to create new array  
var players = ["Amelia", "Emma", "Daisy", "Yvaine"];  
// retrieve value of third player  
var thirdPlayer = players[2];  
// output name of third player  
document.write("<br>Player no.3 = " + thirdPlayer);
```

JavaScript - array access

JS Data Structures - arrays - set, change, add elements

- modify data in an array using a specific index number, e.g.

```
players[3] = "Rose";
```

- updates value in `players` array from Yvaine to Rose
- if we specify an index position beyond the current bounds of the array, e.g.

```
players[5] = "Violet";
```

- array will dynamically expand to add this new value
- index position 4 will now be set to `undefined`
- array's `length` property will also be updated to record new size

JS Data Structures - arrays - set, change, add elements

Modify an array by adding or updating values...

Player no.3 = Rose
New player no.5 = Violet
Player no.4 = undefined

```
var players = ["Amelia", "Emma", "Daisy", "Yvaine"];
// update player name
players[3] = "Rose";
// add new player to the array
players[5] = "Violet";
// output name of third player
document.write("<br>Player no.3 = " + players[3]);
// output new player's name
document.write("<br>New player no.5 = " + players[5]);
// check player 4
document.write("<br>Player no.4 = " + players[4]);
```

JavaScript - array access

JS Data Structures - arrays - set, change, add elements

add new items to array - dynamically expand...

```
> // use array literals to create new array
var players = ["Amelia", "Emma", "Daisy", "Yvaine"];
// update player name
players[3] = "Rose";
// add new player to the array
players[5] = "Violet";
// check updated array
players;
< ▼ (6) ["Amelia", "Emma", "Daisy", "Rose", empty, "Violet"] ⓘ
  0: "Amelia"
  1: "Emma"
  2: "Daisy"
  3: "Rose"
  5: "Violet"
  length: 6
  ► __proto__: Array(0)
```

JavaScript - array access

JS Data Structures - arrays - mix data types

- another benefit of storing data in an array is mixed data types
 - e.g. we *can store numbers with strings...*

```
var players = [1, "Amelia", 42, "Yvaine", "Daisy"];
```

- we can also store an array in an array
 - creates a ***multi-dimensional array***
 - store a number, string, and an inner array

```
var players = [6, "names", ["Amelia", "Emma", "Daisy", "Yvaine", "Rose", "Violet"]];
```

JS Data Structures - arrays - multi-dimensional access

- then access value in an inner array using familiar pattern of index positions, e.g.

```
// create new multi-dimensional array  
var players = [6, "names", ["Amelia", "Emma", "Rose", "Yvaine", "Daisy", "Violet"]  
// get value from inner array - fifth name  
var fifthName = players[2][4];
```


JS Data Structures - arrays - multi-dimensional access

access the inner array of a multi-dimensional array...

fifth name from multi-dimensional array = Daisy

```
> // create new multi-dimensional array
var players = [6, "names", ["Amelia", "Emma", "Rose", "Yvaine", "Daisy", "Violet"]];
// get value from inner array - fifth name
var fifthName = players[2][4];
// output fifth name from multi-dimensional array
document.write("<p>fifth name from multi-dimensional array = " + fifthName + "</p>");
```

JavaScript - array access

fifth name from multi-dimensional array = Daisy

```
> // create new multi-dimensional array
var players = [6, "names", ["Amelia", "Emma", "Rose", "Yvaine", "Daisy", "Violet"]];
// get value from inner array - fifth name
var fifthName = players[2][4];
// output fifth name from multi-dimensional array
document.write("<p>fifth name from multi-dimensional array = " + fifthName + "</p>");
```

JavaScript - array access