Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 3 - Friday

Dr Nick Hayward

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.

- please signup for a CodePen account - https://codepen.io/
  - *use for writing and testing assignment*
  - *send URL to completed PEN for assignment - use private message to TA*

JS Data Structures - Recap - arrays - creating an array

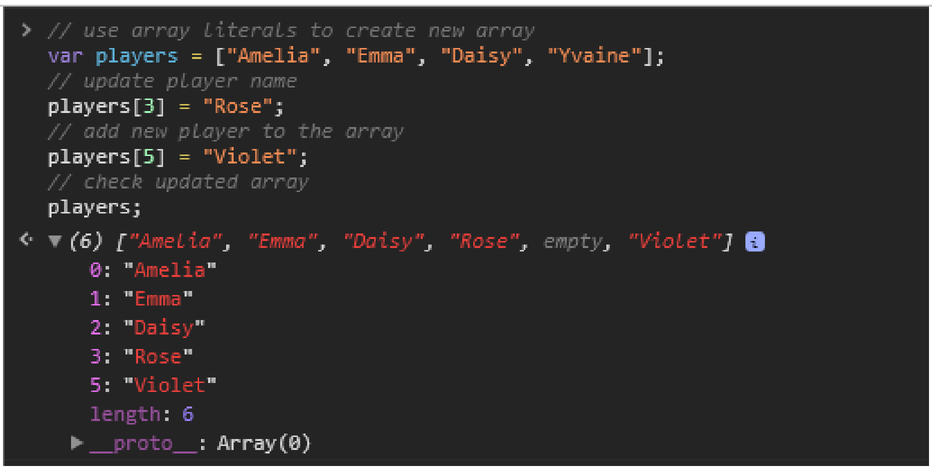
- create an array in JavaScript using two options,
  - *using the built-in Array constructor*
  - *using array literals [ ]*

```
// using array literals to create new array
var players = ["Amelia", "Emma", "Daisy", "Yvaine"];
// using Array constructpr tp 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 - Recap - 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 - Recap - 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"
```

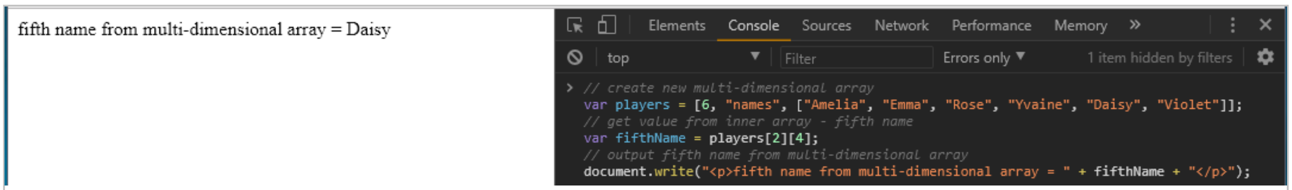
JS Data Structures - Recap - arrays - multi-dimensional access

- 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 - Recap - 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

JS Data Structures - arrays - practical abstraction & usage

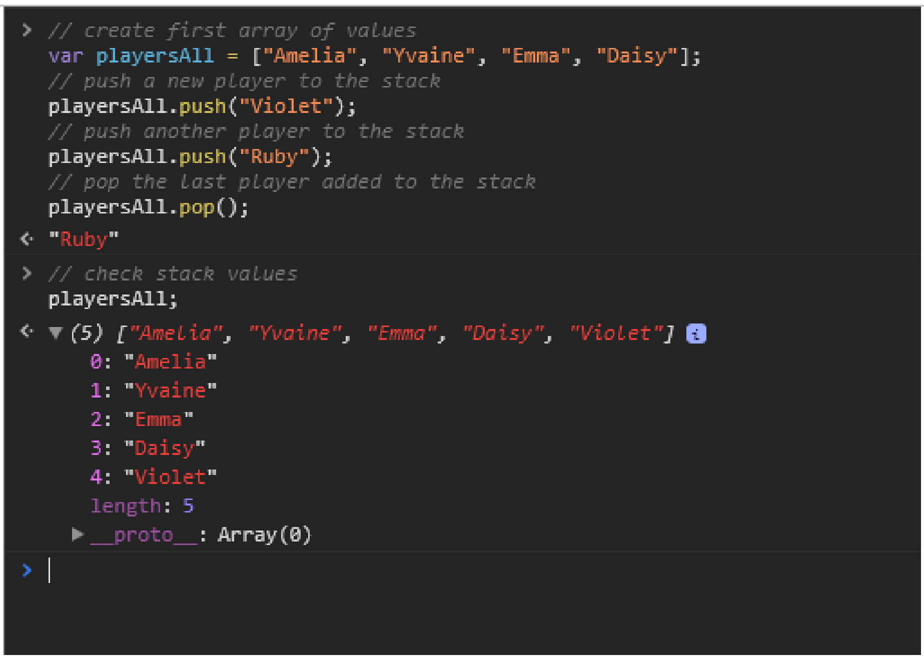
example 1 - create a stack

- many practical uses for an array data structure
- common use is a **stack** to store a sequence of data
- a **stack** stores data in a known, predictable pattern and order
  - *last data in the stack will be the first data out*
- use the following acronym,
  - ***LIFO*** *- Last In, First Out*
  - *use `push( )` and `pop( )` methods to create* ***LIFO****...*

JS Data Structures - arrays - practical abstraction & usage

example 1 - create a stack

use push( ) and pop( ) methods to create **LIFO**...



```
> // create first array of values
  var playersAll = ["Amelia", "Yvaine", "Emma", "Daisy"];
  // push a new player to the stack
  playersAll.push("Violet");
  // push another player to the stack
  playersAll.push("Ruby");
  // pop the last player added to the stack
  playersAll.pop();
<· "Ruby"
> // check stack values
  playersAll;
<· ▼(5) ["Amelia", "Yvaine", "Emma", "Daisy", "Violet"]
      0: "Amelia"
      1: "Yvaine"
      2: "Emma"
      3: "Daisy"
      4: "Violet"
      length: 5
    ▶ __proto__: Array(0)
> |
```

JavaScript - arrays - create a stack

JS Data Structures - arrays - practical abstraction & usage

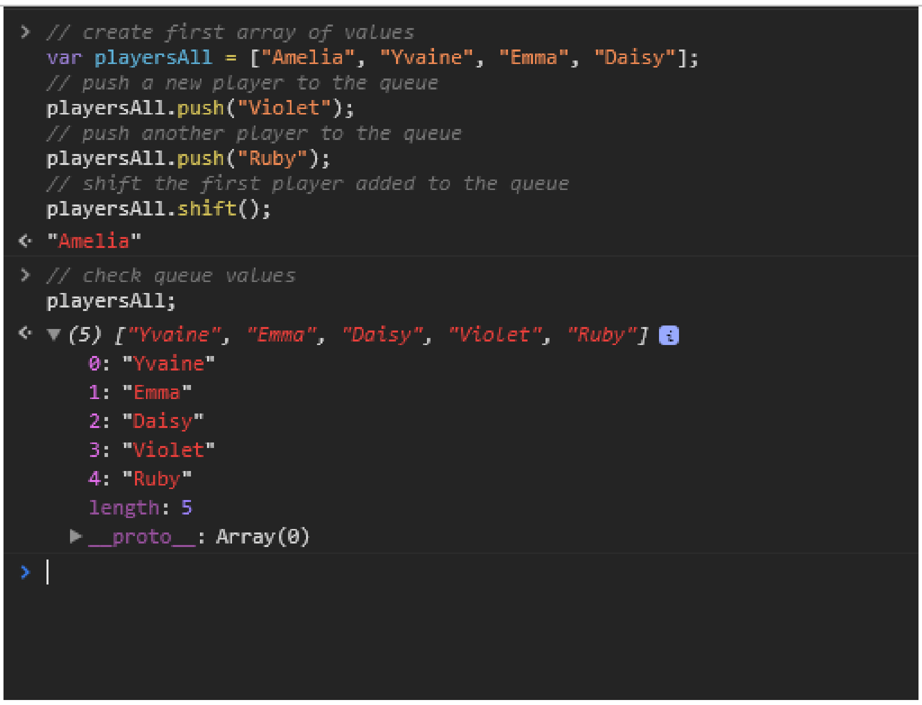
example 2 - create a queue

- also create the opposite of a stack with a **queue**
- like a stack, a **queue** uses a predictable pattern and order
- first data in the queue will be the first data out
  - *use the following acronym,*
  - ***FIFO** - First In, First Out*
- use `push()` and `shift()` methods to create **FIFO**...

JS Data Structures - arrays - practical abstraction & usage

example 2 - create a queue

use `push()` and `shift()` methods to create **FIFO**...



```
> // create first array of values
  var playersAll = ["Amelia", "Yvaine", "Emma", "Daisy"];
  // push a new player to the queue
  playersAll.push("Violet");
  // push another player to the queue
  playersAll.push("Ruby");
  // shift the first player added to the queue
  playersAll.shift();
<· "Amelia"
> // check queue values
  playersAll;
<· ▼(5) ["Yvaine", "Emma", "Daisy", "Violet", "Ruby"] i
      0: "Yvaine"
      1: "Emma"
      2: "Daisy"
      3: "Violet"
      4: "Ruby"
      length: 5
    ▶ __proto__: Array(0)
> |
```

JavaScript - arrays - create a stack

JS Objects - intro

- **object** type includes a compound value
  - *use to set properties, or named locations*
  - *property is an association between **name (or key)** and its value*
  - *`name: value` or `key: value`*
- each of these properties holds its own value
  - *value can be defined as any type*

```
// declear variable - store object literal
var objectA = {
  a: 49,
  b: 59,
  c: "Philae"
};
```

- object literal
  - *curly brackets and everything in between*
- object stores **name:value** (**key:value**) pair/s
  - *quotation marks around property names is optional*
  - *JS knows each name will be string...*
    - quotation marks only needed for multiple words, e.g.

```
var testObject = {
    "Temple Sites": {
        name: "Philae"
    }
}
```

- access these values using either **dot** or **bracket** notation

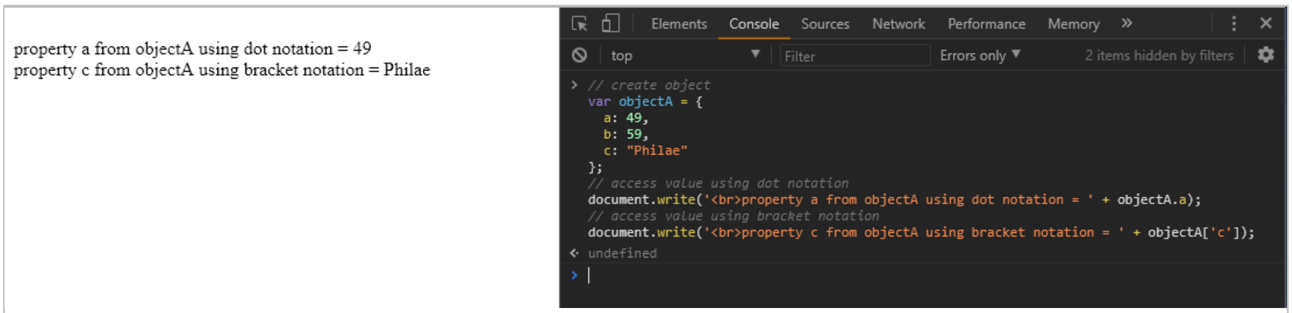
```
//dot notation
objectA.a;
//bracket notation
objectA["a"];
```

JS Objects - object structure

```
|--------------------------------------------|
|            |            |                  |
|            |            |                  |
|   a: 49    |   b: 59    |   c: "Philae"    |
|            |            |                  |
|            |            |                  |
|--------------------------------------------|
```

JS Object structure

JS Objects - example output



```
property a from objectA using dot notation = 49
property c from objectA using bracket notation = Philae
```

```
> // create object
  var objectA = {
    a: 49,
    b: 59,
    c: "Philae"
  };
  // access value using dot notation
  document.write('<br>property a from objectA using dot notation = ' + objectA.a);
  // access value using bracket notation
  document.write('<br>property c from objectA using bracket notation = ' + objectA['c']);
<· undefined
>
```

JS Object - example output

References

- MDN - JavaScript data types and data structure
- W3Schools - Arrays
  - *MDN - Array*