Week 1: JSON/Data Planning

JS Data Types

- Boolean: true/false
- Number: positive, negative, decimals, ints
- String: characters, can use “ “ or ‘ ‘ or ` ` to denote them
- Null
- Undefined: used most commonly when object properties don’t exist
- Array: a list containing any number of elements, can be any type
- Objects: a key/value container

All of the examples on the right use *const*. The syntax is <keyword> <variable name> = <value>, where <keyword> should be *const* for unchanging variables, and *let* for changing variables.

```
// boolean
const bool = true;

// number
const num = 42;

// string
const str = 'Hello World!';

// null
const nul = null;

// undefined
const und = undefined;

// array
const arr = [1, 'Hello World!', true];

// object
const obj = {
    a: 1,
    b: 'Hello World!',
    c: true
};
```

Functions

JS functions are quite similar to other languages but can be written in different ways.

- Basic Named
- Function Expression
- Arrow functions

```
// Basic named
function foo(x, y) {
    return x + y;
}

// Function expression
const foo = function(x, y) {
    return x + y;
}

// Arrow function
const foo = (x, y) => {
    return x + y;
}

// Arrow function with implicit return
const foo = (x, y) => x + y;
```

JSON

JSON is not unique to JS, but is similarly structured to JS objects, and is useful for designing data structures.

To the right is an example of a *collection* of user *documents*. Each user has some properties, like id, name, email, friends, and books.

Database Components

- Collections
  - Usually a list of *documents* with shared properties
    - Every *user* has the same properties: id, name, etc.
- Documents
  - Objects to represent something via particular properties
    - Users, books, animals, bank accounts, assignments
- Functions
  - Actions to view/modify the information stored in the database
    - “getters” will get information from the database
    - “setters” will change information in stored in the database
    - These are all written by you and can be as dynamic as you want

HW: Database Planning

1. Come up with any collections you think your application would need. What groups of information are you going to be storing? *Ex: users, books*

2. What will documents look like in those collections? What properties are they going to need? (hint: it will be easiest to represent this using JSON) *Ex: users need an id, a name, and a list of friends*

3. How will the user be able to interact with your data? Come up with a list of actions your users will be able to take, each of which will end up corresponding to functions. *Ex: viewing a list of all books, changing their email, adding another user as a friend*

HW Example using Trip Planner

1 and 2 shown to the right ->

3:

- create a user, view a user, login, update their information
- create a trip, view all trips for a user, view a trip, update a trip, and delete a trip
- you can create activities, view all activities for a trip, view an activity, update an activity, and delete an activity


```
{
    "users": [
        {
            "username": "user1",
            "password": "password1",
            "trips": ["trip1"]
        },
        {
            "username": "newUser",
            "password": "password2",
            "trips": []
        }
    ],
    "trips": [
        {
            "_id": "trip1",
            "destination": "Portugal",
            "startDate": "2023-01-01",
            "endDate": "2023-01-10",
            "flights": 2,
            "hotel_stays": 1,
            "activities": ["activity1"]
        }
    ],
    "activities": [
        {
            "_id": "activity1",
            "type": "sightseeing",
            "info": "Visit the castle"
        }
    ]
}
```