# CSE 154: Web Programming Spring 2019

## Practice Final Exam 2 (Key)

Name:

UWNet ID: @uw.edu

TA (or section):

**Rules:**

- You have 110 minutes to complete this exam.
- You will receive a deduction if you keep working after the instructor calls for papers.
- You may not use any electronic or computing devices, including calculators, cell phones, smartwatches, and music players.
- Unless otherwise indicated, your code will be graded on proper behavior/output, not on style.
- Do not abbreviate code, such as writing ditto marks () or dot-dot-dot marks (...). You may not use JavaScript frameworks such as jQuery or Prototype when solving problems.
- If you enter the room, you must turn in an exam and will not be permitted to leave without doing so.
- You must show your Student ID to a TA or instructor for your submitted exam to be accepted.

| **Question** | **Score** | **Possible** |
|---|---|---|
| CSS | | 10 |
| Short Answer | | 20 |
| JS/DOM/Timers | | 15 |
| JS/AJAX | | 20 |
| PHP Web Service | | 20 |
| PHP and SQL with PDO | | 15 |
| Extra Credit | | 1 |

## 1. CSS (Cute, Slow, and Sleepy)

```
section {
   margin: auto
   auto; width: 40%;
}

h1 {
   Font-family: Helvetica, Arial,
   sans-serif; text-align: center;
}

div, span {
   border: 1px solid
   black; border-radius:
   50%;
}

span {
   background-color:
   black; display: block;
   Height: 20px;
   width: 20px;
}

#s {
   background-color: sienna;
   height: 150px;
   margin: auto
   auto; width:
   150px;
}

#l {
   align-items: center;
   background-color:
   peru; display: flex;
   height: 110px;
   justify-content:
   space-between; margin-left:
   20px;
   margin-top: 20px;
   width: 110px;
}

#t {
   height: 25px;
}
```

## 2. Short Answer

### a) JavaScript Events

For the following JS program, label the `// __` following each `console.log` statement with 1, 2, 3, or 4, corresponding to the relative order in which that statement will print (where 1 indicates the first statement printed).

```
console.log("Foo");                                 // __1__
(function() {

  console.log("Bar");                               // __2__
  window.onload = pageLoad;
  foo();

  function pageLoad() { console.log("Baz");               // __4__
  }

  function foo() { console.log("Mumble");           // __3__
  }
})();
```

### b) JSON Mystery

Consider the following JSON definition:

```
let mystery = {
  "i" : ["j", 0, 1],
  "ii" : "ii",
  "I" : "i"
};
```

Write the JavaScript value that would be returned for each of the following statements. Include ”” around any string values, and make sure to consider possibly undefined or null values.

| **Statement** | **Return Value** |
|---|---|
| `mystery["i"]` | `[“j”, 0, 1]` |
| `mystery[0]` | `undefined` |
| `mystery.ii.length` | `2` |
| `mystery["i"][0].length` | `1` |

## c) PHP Database Connections

Consider the following PHP code, where `$db` is a defined PDO object:

```
 $str = "INSERT INTO users (username, date, email, password)
         VALUES ('$username', NOW(), '$email', '$password');";
 $rows = $db->exec($str);
```

Briefly explain why this method of using the PDO object is insecure, as well as what change(s) you would need to make it secure given what we've covered in lecture (you may cross out/modify the provided code to indicate the changes).

**Solution:** If the user tries to insert something that has a ’ in the username, email or password, they could potentially do something (at best) to break the query or at worst to inject SQL and do damage to the database. Better would be do do the following:

```
$str = "INSERT INTO users (username, date, email, password)
        VALUES (:username, NOW(), :email, :password);";
$stmt = $db->prepare($str);
$stmt->execute(['username'=>$username, 'email'=>$email, 'password'=>$password]);
$rows = $stmt->fetchAll();
```

## d) Validation Methods

What is one advantage of validating user input on the client (HTML5 or JS) vs. on the server (PHP)?

**Three possible answers:**

• It is faster to validate on the client than on the server

• It can lead to a better UI/UX experience

• It can assist (but not completely solve) the problem of malicious user input by preprocessing the input

What is one advantage of validating user input on the server as opposed to on the client?

**Possible solution:** Validating user input on the server is more secure that validating on the client

## e) Technology Trade-offs: indexDB vs. Dexie

Briefly explain the relationship between indexDB and Dexie and why you might want to use one over the other.

**Possible solutions:**

- Dexie is a wrapper around indexDB that makes it easier to program/use.
- indexDB is more general and broadly supported by more browsers than Dexie.

## f) Web Service Trade-offs: Plain Text vs. JSON

From the client perspective, what is an advantage of working with a JSON response over one in plain text format?

**Possible solutions:**
JSON has indices - easier to parse vs plain text. Maintains structure/hierachy of information for easier understanding to client and general parsing of JSON rather than requiring additional string processing required to get information from plain text.

## g) Storage Technologies

**Solution:** i) localStorage ii) indexDB iii) sessionStorage iv) cookies v) sessions

## h) Regex

Circle all of the following strings which match the regex: **`/^\[-?\d(,\s*-?\d)+\]$/`**

- **`[1, 2, 3]`**
- `[]`
- `[154]`
- `[\d]`
- `0`
- `[-1]`
- `][`
- `[?]`

Circle all of the following strings which match the regex: **`/^<img src=".+.(gif|jpg|png)" alt=".+"\s?\/?>$/i`**

- `^<img src=.+.(gif|jpg|png) alt=.+>$`
- **`<img src="foo.gif" alt="A foo!">`**
- `img src="foo.gif" alt="A foo!"`
- **`<img src="puppy.png" alt="A puppy" />`**
- **`<img src="a.jpg" alt="A letter a" />`**
- **`<img src="FOO.GIF" alt="A foo!">`**

## 4. Plan-It! Fetching You Meals a Day at a Time (20 pts)

```
 (function() {
   "use strict";
   window.addEventListener("click", initialize);

   function initialize() {
     id("day-btn").addEventListener("click", fetchFullMenu);
   }

   function fetchFullMenu() {
     fetch("planit.php?mode=day")
       .then(checkStatus)
       .then(JSON.parse)
       .then(populateFullMenu)
       .catch(console.log);
   }

   function populateFullMenu(responseData) {
     id("day-results").classList.remove("hidden");
     for (let key in responseData) {
       let item = responseData[key];
       qs("#" + key + " .name").innerText = item.name;
       qs("#" + key + " .description").innerText = item.description;
       let ul = qs("#" + key + " .food-groups");
       ul.innerHTML = "";
       for (let i = 0; i < item["food-groups"].length; i++) {
            let li = document.createElement("li");
            li.innerText = item["food-groups"][i];
            ul.appendChild(li);
       }
     }
   }
})();
```

## 5. *Serving* Up Some Meal Ideas (20 pts)

### Part A:

**Solution:**

```
function get_meal_data($meal) {
    $choices = glob("{$meal}/*.txt");
    $r = $choices[array_rand($choices)];
    $data = file($r, FILE_IGNORE_NEW_LINES);
    return array("name" => $data[0],
                 "description" => $data[1],
                 "food-groups" => explode(" ", $data[2]));

}
```

### Part B:

**Solution:**

```
<?php
if (isset($_GET["mode"])) {
  $mode = strtolower($_GET["mode"]);

  if ($mode === "day") {
    $result = array();
    $result["breakfast"] = get_meal_data("breakfast");
    $result["lunch"] = get_meal_data("lunch");
    $result["dinner"] = get_meal_data("dinner");
    header("Content-type: application/json");
    print(json_encode($result));

  } else {
    header("Content-type: text/plain");
    if ($mode === "meal") {
      if (isset($_GET["type"])) {
        $type = strtolower($_GET["type"]);
        $choices = glob("{$type}/*.txt");
        for ($i = 0; $i < count($choices) - 1; $i++) {
        $lines = file($choices[$i], FILE_IGNORE_NEW_LINES);
        print ("{$lines[0]}: {$lines[1]}\n");
      }
      $lines = file($choices[count($choices) - 1], FILE_IGNORE_NEW_LINES);

      print ("{$lines[0]}: {$lines[1]}");
    } else { # missing type
      header("HTTP 1.1 400 Invalid Request");
      die("Missing type parameter for meal request");
    }
  } else { # incorrect mode
    header("HTTP/1.1 400 Invalid Request");
    die("Please pass a mode of meal or day");
  }
} else { # missing mode
  header("Content-type: text/plain");
  header("HTTP/1.1 400 Invalid Request");
  die("Please pass a mode of meal or day");
}
?>
```

## 6. PHP and SQL Connection with PDO

**Part A:**

**i.**

```
SELECT id, name, platform FROM games WHERE platform LIKE “PS%”;
```

**ii.**

```
INSERT INTO games (name, platform, release_year, genre, publisher)
VALUES (“Baba is You”, “PC”, 2018, “Puzzle”, “Hempuli Oy”);
```

**iii.**

```
UPDATE games SET release_year=”2019” WHERE id=3001;
```

**Part B:**

**i.**

**d.** `“DELETE FROM games WHERE genre = :genre;”`

**ii.**

```
function delete_games_with_genre($genre) {
  // you may assume that get_PDO returns the appropriate PDO object
  $db = get_PDO();
  try {
    $sql = "DELETE FROM games WHERE genre = :genre;";
    // TODO: Use $db with the $sql string to securely execute the query,
    // and define $count so that the success message below includes the
    // number of rows removed from the table.

    $stmt = $db->prepare($sql);
    $params = array("genre" => $genre);
    $stmt->execute($params);
    $count = $stmt->rowCount();

    // You may assume that output_success correctly sets any needed headers
    // and outputs the message passed to it.
    output_success("Successfully deleted {$count} games from the table!");
  }
  catch (PDOException $ex) {
      // You may assume that the below function sets the correct
      // headers, outputs the message, and stops the script.
      handle_db_error("Can not connect to the database. Please try again later.");
  }
}
```

## X. Extra Credit