Haamed Azimi

JS Fundamentals Lab Exercise

Questions:

1. **What are the results of these expressions? (answer first, then use console.log() to check)**

    1. " " + 1 + 0 **My Answer: The string: "(space)10"** *Answer after using console.log(): "(space)10"*
    2. " " - 1 + 0 **My Answer: The string: "(space) -1"** *Answer after using console.log(): -1*
    3. true + false **My Answer: Integer: 1** *Answer after using console.log(): Integer: 1*
    4. !true **My Answer: false** *Answer after using console.log(): false*
    5. 6 / "3" **My Answer: The Integer 2** *Answer after using console.log():* **The Integer 2**
    6. "2" * "3" **My Answer: The Integer 6** *Answer after using console.log():* **The Integer 6**
    7. 4 + 5 + "px" **My Answer: The String "9px"** *Answer after using console.log():* **The String "9px"**
    8. "$" + 4 + 5 **My Answer: The String "$45"** *Answer after using console.log():* **The String "$45"**
    9. "4" - 2 **My Answer: Integer: 2** *Answer after using console.log(): Integer: 2*
    10. "4px" - 2 **My Answer: Throws an error** *Answer after using console.log(): Nan*
    11. " -9 " + 5 **My Answer: Integer: -4** *Answer after using console.log(): String: -9 5*
    12. " -9 " - 5 **My Answer: Nan** *Answer after using console.log(): -14*
    13. null + 1 **My Answer: Error** *Answer after using console.log(): 1*
    14. undefined + 1 **My Answer: Error** *Answer after using console.log(): Nan*
    15. undefined == null **My Answer: false** *Answer after using console.log(): true*
    16. undefined === null **My Answer: false** *Answer after using console.log(): false*
    17. " \t \n" - 2 **My Answer: String: "(one tab) (on the next line) -2"** *Answer after using console.log(): Integer: -2*

2. Which of the below are not giving the right answer? Why are they not correct? How can we fix them?

```
let three = "3"
let four = "4"
let thirty = "30"
//what is the value of the following expressions?
let addition = three + four
let multiplication = three * four
let division = three / four
let subtraction = three - four
let lessThan1 = three < four
let lessThan2 = thirty < four
```

Answers:

- let addition = three + four **will be "34" (34 as a string)** because it concatenates the two strings, but if we want the addition to happen we need to convert the string to numbers.
- let multiplication = three * four **will be 12 (12 as an integer)** This probably works as intended because Javascript converts them for us when it comes to multiplication.
- let division = three / four will be **0.75 (0.75 as a number)** The same goes here, Javascript handles the type conversion for use.
- let subtraction = three - four **will be -1 (-1 as an integer)** Here also javascript converts the string to numbers.

- let lessThan1 = three < four **will be true** This will return true because Javascript compares both of the strings based on their Unicode. The Unicode for “3” is U+0033 and for “4” the Unicode is U+0034 so 33 is less than 34 that’s why it returns true.
- let lessThan2 = thirty < four **will be false** Here also "4" comes after "30" because the first character "4" (U+0034) is greater than the first character "3" (U+0033).

3. Which of the following console.log messages will print? Why?

```
if (0) console.log('#1 zero is true')
if ("0") console.log('#2 zero is true')
if (null) console.log('null is true')
if (-1) console.log('negative is true')
if (1) console.log('positive is true’)
```

Answers:

if (0) console.log('#1 zero is true’) won’t be logged in the console because 0 is falsy in Javascript.
if ("0") console.log('#2 zero is true’) will be logged in the console because “0” is not an empty string and non empty strings are considered truthy in Javascript.
if (null) console.log('null is true') won’t be logged in the console because null is considered as a falsy value in Javascript so the condition evaluates to false.
if (-1) console.log('negative is true’) will get logged to the console because -1 is a non zero number and all numbers but zero is considered true in javascript.
if (1) console.log('positive is true’) will also get logged to the console as it’s considered as a truthy value so the condition will evaluate to true.

4. Rewrite this `if` using the ternary/conditional operator '`?`'. Test it with different values for `a` and `b`. What does the ‘+=’ do?

```
let a = 2, b = 3;
let result = `${a} + ${b} is `;

if (a + b < 10) {
result += 'less than 10';
} else {
result += 'greater than 10';
}
```

Answers:

Example 1:

```
let a = 4, b = 5;
let result = a + b < 10 ? `${a} + ${b} is less than 10` : `${a} + ${b} is greater than 10`;
console.log(result);          This will output: "4 + 5 is less than 10”
```

Example 2:

```
let a = 7, b = 6;
let result = a + b < 10 ? `${a} + ${b} is less than 10` : `${a} + ${b} is greater than 10`;
console.log(result);          This will output: "7 + 6 is greater than 10"
```

Example 3: If they are both equal we can add another condition to handle that.

```
let a = 5, b = 5;
let result = a + b < 10 ? `${a} + ${b} is less than 10`
        : a + b === 10 ? `${a} + ${b} is equal to 10`
        : `${a} + ${b} is greater than 10`;
console.log(result); // Output: "5 + 5 is equal to 10"
```

5. Rewrite the following function using: a) function expression syntax, and b) arrow function syntax. Test each version to make sure they work the same.

```
function getGreeting(name) {
return 'Hello ' + name + '!';
}
```

Answers:

let getGreeting = (name) => 'Hello ' + name + '!';

Console.log(getGreeting("Haamed"));

I used both types of the syntax and got the same result.

6. a) Complete the `inigo` object by adding a `lastName` property and including it in the `greeting`.
b) Complete `getCatchPhrase` so that if the `person` argument has 6 fingers, it instead prints his famous catch phrase to the console. HINT: see https://www.imdb.com/title/tt0093779/characters/nm0001597.
c) Update `getCatchPhrase` to use arrow function syntax and a conditional operator.

```
const westley = {
name: 'Westley',
numFingers: 5
}
const rugen = {
name: 'Count Rugen',
numFingers: 6
}
const inigo = {
firstName: 'Inigo',
greeting(person) {
let greeting = `Hello ${person.name}, my name is ${this.firstName}. `;
console.log(greeting + this.getCatchPhrase(person));
},
getCatchPhrase(person) {
return 'Nice to meet you.';
}
}
inigo.greeting(westley)
inigo.greeting(rugen)
```

Answers:

```
const westley = {
  name: "Westley",
  numFingers: 5,
};
const rugen = {
  name: "Count Rugen",
  numFingers: 6,
};
const inigo = {
  firstName: "Inigo",
  lastName : "Montoya",
  greeting(person) {
    let greeting = `Hello ${person.name}, my name is ${this.firstName} ${this.lastName}. `;
    console.log(greeting + this.getCatchPhrase(person));
  },
  getCatchPhrase : (person) => {

    return person.numFingers >= 6 ? "You killed my father. Prepare to die." : "Nice to meet you.";
  },
};
inigo.greeting(westley);
inigo.greeting(rugen);
```

7. The following object represents a basketball game and keeps track of the score as the game progresses.
a) Modify each of the methods so that they can be 'chained' together and the last line of the example code works
b) Add a new method to print the full time final score
c) Add a new object property to keep track of the number of fouls and a method to increment it, similar but separate to the score. Include the foul count in the half time and full time console messages
d) Test your object by chaining all the method calls together in different combinations.

```
const basketballGame = {
score: 0,
freeThrow() {
this.score++;
},
basket() {
this.score += 2;
},
threePointer() {
this.score += 3;
```

```
},
halfTime() {
console.log('Halftime score is '+this.score);
}
}
//modify each of the above object methods to enable function chaining as below:
basketballGame.basket().freeThrow().freeThrow().basket().threePointer().halfTime();
```

Answers:

```
function basketballGame() {
  this.score = 0;
  this.halfTimeUsed = false;
  this.foul = 0;
}

basketballGame.prototype.freeThrow = function () {
  this.score++;
  return this;
};

basketballGame.prototype.basket = function () {
  this.score += 2;
  return this;
};

basketballGame.prototype.threePointer = function () {
  this.score += 3;
  return this;
};

basketballGame.prototype.fouled = function () {
  this.foul++;
  console.log(`Foul!! Foul Count: ${this.foul}`);
  return this;
};

basketballGame.prototype.halfTime = function () {
  if (this.halfTimeUsed) {
    console.log("There can only be one half time per game;");
    return this;
  }

  console.log("Halftime score is " + this.score);
  this.halfTimeUsed = true;
  return this;
};

basketballGame.prototype.fullTime = function () {
  if (this.halfTimeUsed) {
    console.log("Full TIme score is " + this.score);
    return this;
  }

  console.log(
    `It's not full time yet, wait for half time then we'll have the full time score. \nRight now the score is $
{this.score}`
  );
  return this;
};
```

```
//modify each of the above object methods to enable function chaining as below:
console.log("First Game:")
let firstGame = new basketballGame()
  .basket()
  .freeThrow()
  .freeThrow()
  .basket()
  .threePointer()
  .halfTime()
  .fouled()
  .fullTime();


  console.log("Second Game:")
  let secondGame = new basketballGame()

  .threePointer()
  .freeThrow()
  .fouled()
  .basket()
  .halfTime()
  .fouled()
  .threePointer()
  .fullTime();
```

8. The object below represents a single city.
a) Write a function that takes an object as an argument and uses a `for...in` loop to access and print to the console each of those object properties and their values. Test it using the `sydney` object below.
b) Create a new object for a different city with different properties and call your function again with the new object.

```
const sydney = {
name: 'Sydney',
population: 5_121_000,
state: 'NSW',
founded: '26 January 1788',
timezone: 'Australia/Sydney'
}
```

Answers:

```
const sydney = {
name: 'Sydney',
population: 5_121_000,
state: 'NSW',
founded: '26 January 1788',
timezone: 'Australia/Sydney'
};


const perth = {
  name: 'Perth',
  population: 7_189_000,
  state: 'WA',
  founded: '1929',
  timezone: 'Australia/Perth'
  };
```

```
const objectPropVal = (obj) =>{
  for (const key in obj) {
     console.log(`${key} : ${obj[key]}`)
  }
}

objectPropVal(sydney);
console.log("------------");
objectPropVal(perth);
```

9. Use the following variables to understand how JavaScript stores objects by reference.
a) Create a new `moreSports` variable equal to `teamSports` and add some new sport values to it (using `push` and `unshift`)
b) Create a new `dog2` variable equal to `dog1` and give it a new value
c) Create a new `cat2` variable equal to `cat1` and change its `name` property
d) Print the original `teamSports`, `dog1` and `cat1` variables to the console. Have they changed? Why?
e) Change the way the `moreSports` and `cat2` variables are created to ensure the originals remain independent

```
let teamSports = ['Hockey', 'Cricket', 'Volleyball'];
let dog1 = 'Bingo';
let cat1 = { name: 'Fluffy', breed: 'Siberian' };
```

Answers:
I have changed the code by using classes as they can be used to create multiple copies of the same object without disrupting the other one. Also, I have used a Dogs class and a Cats class for separately as that's what the task had asked for, else we could use an Animal class which is a more generic one.

```
class ListSports {
  constructor(listOfSports) {
    this.listOfSports = listOfSports;
  }

  printSportList() {
    console.log(this.listOfSports);
  }
}

class Dogs {
  constructor(dogName) {
    this.dogName = dogName;
  }

  printDogName() {
    console.log(this.dogName);
  }
}

class cats {
  constructor({ name, breed }) {
    this.name = name;
    this.breed = breed;
  }

  printCatDetails() {
```

```
    console.log(this);
  }
}

let teamSports = new ListSports(["Hockey", "Cricket", "Volleyball"]);
teamSports.printSportList();

console.log("----------------");

let dog1 = new Dogs("Tony");
dog1.printDogName();

console.log("----------------");
let cat1 = new cats({ name: "Fluffy", breed: "Siberian" });
cat1.printCatDetails();
```

10. The following constructor function creates a new `Person` object with the given `name` and `age` values.
a) Create a new person using the constructor function and store it in a variable.
b) Create a second person using different `name` and `age` values and store it in a separate variable.
c) Print out the properties of each person object to the console
d) Rewrite the constructor function as a class called `PersonClass` and use it to create a third person using different name and age values. Print it to the console as well.
e) Add a `canDrive` method to both the constructor function and the class that returns true if the person is old enough to drive.

```
function Person(name, age) {
this.name = name;
this.age = age;
this.human = true;
}
```

Answers:

```
function Person(name, age) {
  this.name = name;
  this.age = age;
  this.human = true;
}

Person.prototype.canDrive = function () {
  return this.age >= 18 ? true : false;
};

let person1 = new Person("Tony", 19);
let person2 = new Person("Ethan", 34);

console.log(person1.name);
console.log(person1.age);
console.log(person1.canDrive());
console.log("----------");
console.log(person2.name);
console.log(person2.age);
console.log(person2.canDrive());
```

```
console.log("----------");

class PersonClass {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  canDrive() {
    return this.age >= 18 ? true : false;
  }
}

let person3 = new PersonClass("Smith", 52);
console.log(person3.name);
console.log(person3.age);
console.log(person3.canDrive());
```