Code Requirement List

1. Implement the following methods of **Lodash**.

    - hunck()
    - compact()
    - concat()
    - differenceWith()
    - drop()
    - fromPairs()
    - intersection()
    - zip()
    - unzip()
    - xor()
    - groupBy()
    - orderBy()
    - reject()
    - some()
    - sortBy()

2. Write a function that inserts a white space between every instance of a lower character followed immediately by an upper character.

Input	Output
insertWhitespace("SheWalksToTheBeach")	"She Walks To The Beach"
insertWhitespace("MarvinTalksTooMuch")	"Marvin Talks Too Much"
insertWhitespace("TheGreatestUpsetInHistory")	"The Greatest Upset In History"

3. In JavaScript ES6 an arrow function expression is a syntactically compact alternative to a regular function expression.

    Create a function that takes a string representing a function and converts between an arrow function and a regular function.

    - If the input is a regular function, return an equivalent arrow function.
    - If the input is a arrow function, return an equivalent regular function.

Input	Output
"function () {}"	"() => {}"
"function name() {}"	"const name = () => {}"
"function name(str) { console.log(str); }"	"const name = (str) => { console.log(str); }"
"() => {}"	"function () {}"

4. Each year has 365 or 366 days. Given a string date representing a Gregorian calendar date formatted as month/day/year, return the day-number of the year. *All input strings in the tests are valid dates.*

Input	Output
dayOfYear("12/13/2020")	348
dayOfYear("12/17/2020")	352
dayOfYear("11/16/2020")	321
dayOfYear("1/9/2019")	9

5. Write a function that, given a date (in the format MM/DD/YYYY), returns the day of the week as a string. Each day name must be one of the following strings: "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", or "Saturday".

   To illustrate, the day of the week for "12/07/2016" is "Wednesday".

   - This challenge assumes the week starts on Sunday.*

Input	Output
getDay("12/07/2016")	"Wednesday"

getDay("09/04/2016")	"Sunday"
getDay("12/08/2011")	"Thursday"

- Implement following function

```
const formatDate = (date) => {
  return date;
};
console.log("Actual output: ", formatDate(new Date("2020-05-14")));
console.log("Expected output", "14 May 2020");
```

- Implement following function

```
const getWeekOfMonth = () => {};
const result = getWeekOfMonth(new Date(2017, 10, 9));
// => 2
```

6. Create a function that implements partial sum.

```
sum(1)(2); // 3
const addOne = sum(1);
addOne(2); // 3
const addTen = sum(10);
addTen(2); // 12
addOne(4); // 5
addTen(10); // 20
```

7. Write a function `redundant` that takes in a string `str` and returns a function that returns `str`.

```
const f1 = redundant("apple");
```

```
//f1() → "apple"


const f2 = redundant("pear");

//f2() → "pear"


const f3 = redundant("");

//f3() → ""
```

8. Write a function that returns an anonymous function, which transforms its input by adding a particular suffix at the end.

```
add_ly = add_suffix("ly");


add_ly("hopeless"); // "hopelessly"

add_ly("total"); // "totally"


add_less = add_suffix("less");


add_less("fear"); // "fearless"

add_less("ruth"); // "ruthless"
```

9. Create a function `printAfter` that calls its argument after printing `hello world`

```
 const print = () => console.log("Elon Musk");

 printAfter(print);

 // 'hello, world'

 // 'Elon Musk'
```

10. Get name, country and jod using destructuring

```
const person = { name: "Sarah", country: "Armenia", job: "Developer" };


console.log(name); // "Sarah"

console.log(country); // "Nigeria"

console.log(job); // "Developer"
```

11. Concatenate the two arrays

```
const arr1 = [1, 2, 3, 4];

const arr2 = [5, 6, 7, 8, 9];


const arr3; // [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

12. How to take arguments in sum function?

```
function sum() {
  return args.reduce((sum, current) => {
    return sum + current;
  });
}
sum(1, 2); // prints 3
sum(1, 2, 3); // prints 6
```

13. How to pass arguments in sum function?

```
function sum(x, y, z) {
  return x + y + z;
}
```

```
const numbers = [1, 2, 3];

console.log(sum()); // prints 6
```

14. Swapping Values using the Destructuring Assignment

```
let a = 3;

let b = 6;

console.log(a); //6

console.log(b); //3
```

15. Make a Circle with methods

    Your task is to create a Circle constructor that creates a circle with a radius provided by an argument. The circles constructed must have two methods getArea() (PI*r^2) and getPerimeter() (2*PI*r) which give both respective areas and perimeter (circumference).

    For help with this class, I have provided you with a Rectangle constructor which you can use as a base example.

```
const circy = new Circle(11);

circy.getArea();

// Should return 380.132711084365


const circy = new Circle(4.44);

circy.getPerimeter();

// Should return 27.897342763877365
```

16. Write a function that creates an object with each (key, value) pair being the (lower case, upper case) versions of a letter, respectively.

```
mapping(["p", "s"]) → { "p": "P", "s": "S" }

mapping(["a", "b", "c"]) → { "a": "A", "b": "B", "c": "C" }
```

```
mapping(["a", "v", "y", "z"]) → { "a": "A", "v": "V", "y": "Y", "z": "Z"
}
```

17. Create a function that returns the frequency distribution of an array. This function should return an object, where the keys are the unique elements and the values are the frequency in which those elements occur.

```
getFrequencies(["A", "B", "A", "A", "A"]) → { A: 4, B: 1 }

getFrequencies([1, 2, 3, 3, 2]) → { "1": 1, "2": 2, "3": 2 }

getFrequencies([true, false, true, false, false]) → { true: 2, false: 3 }

getFrequencies([]) → {}
```

18. You receive an `object` with nested objects with `strings` as `values`. Convert their values to `number` and return an `object` without the entries that evaluate to `NaN`.

```
findAndRemove({

  bedroom: {

    slippers: "10000",

    piano: "550",

    call: "vet",

    travel: "world",

  },

});

// => OUTPUT

// {

//   bedroom: {

//     slippers: 10000,

//     piano: 5500,

//   },

// }
```

19. Create a function that takes a mathematical expression in **prefix notation** as a string and evaluates the expression.

```
prefix("+ 5 4") → 9

prefix("* 12 2") → 24

prefix("+ -10 10") → 0
```

20. Your task is to create a function that simulates a vending machine.

    Given an amount of money (in cents ¢ to make it simpler) and a productNumber, the vending machine should output the correct product name and give back the correct amount of change.

    The coins used for the change are the following: [500, 200, 100, 50, 20, 10]

The return value is an object with 2 properties:

- product: the product name that the user selected.
- change: an array of coins (can be empty, must be sorted in descending order).

vendingMachine(products, 200, 7) → { product: "Crackers", change: [50, 20, 10]

vendingMachine(products, 500, 0) → "Enter a valid product number"

vendingMachine(products, 90, 1) → "Not enough money for this product"

Notes

- The products are fixed and you can find them in the **Tests** tab.
- If productNumber is invalid (out of range) you should return the *string*: "Enter a valid product number".
- If money is not enough to buy a certain product you should return the *string*: "Not enough money for this product".
- If there's no change, return an empty array as the change.

```
const products = [
  { number: 1, price: 100, name: 'Orange juice' },
  { number: 2, price: 200, name: 'Soda' },
  { number: 3, price: 150, name: 'Chocolate snack' },
  { number: 4, price: 250, name: 'Cookies' },
  { number: 5, price: 180, name: 'Gummy bears' },
  { number: 6, price: 500, name: 'Condoms' },
  { number: 7, price: 120, name: 'Crackers' },
  { number: 8, price: 220, name: 'Potato chips' },
  { number: 9, price: 80,  name: 'Small snack' },
];
```