1.

When you divide the successive powers of 10 by 13 you get the following remainders of the integer divisions:

1, 10, 9, 12, 3, 4.

Then the whole pattern repeats.

Hence the following method: Multiply the right most digit of the number with the left most number in the sequence shown above, the second right most digit to the second left most digit of the number in the sequence. The cycle goes on and you sum all these products. Repeat this process until the sequence of sums is stationary.

...........................................................................

Example: What is the remainder when 1234567 is divided by 13?

7×1 + 6×10 + 5×9 + 4×12 + 3×3 + 2×4 + 1×1 = 178

We repeat the process with 178:

8x1 + 7x10 + 1x9 = 87

and again with 87:

7x1 + 8x10 = 87

...........................................................................

From now on the sequence is stationary and the remainder of 1234567 by 13 is the same as the remainder of 87 by 13: 9

Call thirt the function which processes this sequence of operations on an integer n (>=0). thirt will return the stationary number.

thirt(1234567) calculates 178, then 87, then 87 and returns 87.

thirt(321) calculates 48, 48 and returns 48

Answer

Mysolution

function thirt(n) {

  let pattern = [1, 10, 9, 12, 3, 4];

  let reversedNumber = (n + "").split("").reverse();

  let size = 0;

  let total = 0;

  for (let index = 0; index < reversedNumber.length; index++) {

    if (size == 6) {

      size = 0;

    }

    total += pattern[size] \* parseInt(reversedNumber[index]);

    size++;

  }

  if (total < 100) {

    return total;

  } else {

    return thirt(total);

  }

}

Clever solution

function thirt(n) {

  const nums = [1, 10, 9, 12, 3, 4];

  var sum = ("" + n)

    .split("")

    .reverse()

    .reduce((sum, v, i) => {

      return sum + v \* nums[i % nums.length];

    }, 0);

  return sum === n ? n : thirt(sum);

}

Patta pattern ekak

\* nums[i % nums.length];

Eg-: Array(21)

  .fill(1)

  .forEach((element, index) => {

    //5 dammoth 0123401234 repeate wenawa

    console.log(index);

    console.log(index % 5);

  });

output

0

1

2

3

4

0

1

2

3

4

0

1

2

3

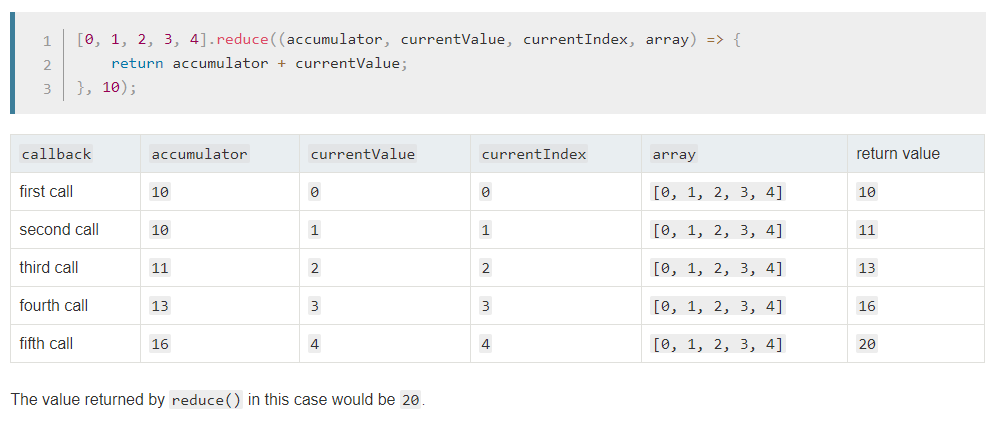
4

0

1

2

\*reduce behavior



Write a function defaultArguments. It takes a function as an argument, along with an object containing default values for that function's arguments, and returns another function which defaults to the right values.

You cannot assume that the function's arguments have any particular names.

You should be able to call defaultArguments repeatedly to change the defaults.

function add(a,b) { return a+b;};

var add\_ = defaultArguments(add,{b:9});

add\_(10); // returns 19

add\_(10,7); // returns 17

add\_(); // returns NaN

add\_ = defaultArguments(add\_,{b:3, a:2});

add\_(10); // returns 13 now

add\_(); // returns 5

add\_ = defaultArguments(add\_,{c:3}); // doesn't do anything, since c isn't an argument

add\_(10); // returns NaN

add\_(10,10); // returns 20

function defaultArguments(func, params) {

  let args = func

    .toString()

    .replace(/(\r\n|\n|\r)/gm, "")

    .match(/\((.\*)\)/)[1]

    .split(",")

    .map(e => e.trim());

  args = args.map(p => {

    p = p.replace(/=.\*/, "");

    if (params[p]) {

      return `${p}=${params[p]}`;

    }

    return p;

  });

  return new Function(args, func.toString().match(/{([\s\S])\*/g)[0]);

}

function add(a, b) {

  return a + b;

}

var add\_ = defaultArguments(add, { b: 9 });

add\_(10);

add\_ = defaultArguments(add\_, { b: 3 });

console.log(add\_(10));

**top\_3\_words("In a village of La Mancha, the name of which I have no desire to call to**

**mind, there lived not long since one of those gentlemen that keep a lance**

**in the lance-rack, an old buckler, a lean hack, and a greyhound for**

**coursing. An olla of rather more beef than mutton, a salad on most**

**nights, scraps on Saturdays, lentils on Fridays, and a pigeon or so extra**

**on Sundays, made away with three-quarters of his income.")**

**# => ["a", "of", "on"]**

**top\_3\_words("e e e e DDD ddd DdD: ddd ddd aa aA Aa, bb cc cC e e e")**

**# => ["e", "ddd", "aa"]**

**top\_3\_words(" //wont won't won't")**

**# => ["won't", "wont"]**

Mycode

function topThreeWords(text) {

  const map = new Map();

  if (text.trim().length === 1) {

    return [];

  }

  text

    .toLowerCase()

    .match(/[\w']+/g)

    .forEach(element => {

      return map.has(element)

        ? map.set(element, map.get(element) + 1)

        : map.set(element, 1);

    });

  return [...map.entries()]

    .sort((a, b) => {

      return b[1] - a[1];

    })

    .slice(0, 3)

    .map(e => {

      return e[0];

    });

}

Meka hodata liwwa ekak

alexshavlovsky

// 07.09.2018

let topThreeWords = text => {

    let dict = new Map();

    text.replace(/[A-z']+(?=[ ]+|$)/g, match => {

        let word = match.toLowerCase();

        dict.set(word, dict.has(word) ? dict.get(word) + 1 : 1);

    });

    dict.delete("'");

    return [...dict].sort((a, b) => b[1] - a[1]).map(a => a[0]).slice(0, 3);

};

**In this kata we want to convert a string into an integer. The strings simply represent the numbers in words.**

**Examples:**

* **"one" => 1**
* **"twenty" => 20**
* **"two hundred forty-six" => 246**
* **"seven hundred eighty-three thousand nine hundred and nineteen" => 783919**

**Additional Notes:**

* **The minimum number is "zero" (inclusively)**
* **The maximum number, which must be supported is 1 million (inclusively)**
* **The "and" in e.g. "one hundred and twenty-four" is optional, in some cases it's present and in others it's not**
* **All tested numbers are valid, you don't need to validate them**

function parseInt(string) {

  let data = new Map([

    ["one", "+1"],

    ["two", "+2"],

    ["three", "+3"],

    ["four", "+4"],

    ["five", "+5"],

    ["six", "+6"],

    ["seven", "+7"],

    ["eight", "+8"],

    ["nine", "+9"],

    ["ten", "+10"],

    ["zero", 0],

    ["eleven", 11],

    ["twelve", 12],

    ["thirteen", 13],

    ["fourteen", 14],

    ["fifteen", 15],

    ["sixteen", 16],

    ["seventeen", 17],

    ["eighteen", 18],

    ["nineteen", 19],

    ["twenty", 20],

    ["thirty", 30],

    ["forty", 40],

    ["fifty", 50],

    ["sixty", 60],

    ["seventy", 70],

    ["eighty", 80],

    ["ninety", 90],

    ["hundred", "\*100+"],

    ["thousand", "\*1000+"],

    ["million", "\*1000000+"],

    ["and", ""]

  ]);

  return eval(

    string

      .replace(/\w+/g, params => {

        return data.get(params);

      })

      .replace(/[-| ]/g, "")

      .trim()

      .replace(/((.+)\\*(1000))/g, "($2)\*$3")

      .replace(/\+\+/g, "+")

      .replace(/(\+)$/, "")

      .replace(/\+([^\d])/, "$1")

  );

}

console.log(parseInt("one million"));

//another question

//function replacer(match, p1, p2, p3, offset, string)

// I've seen the underscore used to denote that the variable is a "don't care" variable.

// It means that it doesn't matter and isn't used at all.

// In the case you pointed out, it was used to signify that his function had two arguments,

// but he only needed the second one.

function toCamelCase(str) {

  return str.replace(/[-\_](\w)/g, (\_, i) => {

    return i.toUpperCase();

  });

}

console.log(toCamelCase("the\_stealth\_warrior"));

function toCamelCase1(str) {

  return str.replace(/[-\_](\w)/g, (...e) => {

    return e[1].toUpperCase();

  });

}