RegEx in JavaScript

The Class of Characters

You have 26 alphabets:
a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z

RegEx provides a way to capture a character from the above class of alphabets.

Let’s do it through code...

Checking if a given character is alphabetical or not

```
let c = '5'; // Simple variable declaration. Variable of type: “string”

let alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
// Another simple variable declaration. Variable of type: Array

let msg = 'C IS NOT AN ALPHABET'
// ‘msg’ is of type: String

for (let i = 0; i < alphabets.length; i++) {
    if (c == alphabets[i]){
        msg = "C IS AN ALPHABET"
    }
}

// To iterate over ‘alphabets’ we need a variable called ‘i’.
// i starts taking values from 0 till alphabets.length (which is 26)
// Next in the if-condition ( c == alphabets[i] ): it is comparing each alphabet with the contents of variable ‘c’

console.log(msg)
```

```
» let c = '5'; // Simple variable declaration. Variable of type: “string”

  let alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
  // Another simple variable declaration. Variable of type: Array

  let msg = 'C IS NOT AN ALPHABET'
  // ‘msg’ is of type: String

  for (let i = 0; i < alphabets.length; i++) {
      if (c == alphabets[i]){
          msg = "C IS AN ALPHABET"
      }
  }

  // To iterate over ‘alphabets’ we need a variable called ‘i’.
  // i starts taking values from 0 till alphabets.length (which is 26)
  // Next in the if-condition ( c == alphabets[i] ): it is comparing each

  console.log(msg)
C IS NOT AN ALPHABET
```

```
let c = 'b'; // Simple variable declaration. Variable of type: “string”

let alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n'
// Another simple variable declaration. Variable of type: Array

let msg = 'C IS NOT AN ALPHABET'
// ‘msg’ is of type: String

for (let i = 0; i < alphabets.length; i++) {
    if (c == alphabets[i]){
        msg = "C IS AN ALPHABET"
    }
}

// To iterate over ‘alphabets’ we need a variable called ‘i’.
// i starts taking values from 0 till alphabets.length (which is 26)
// Next in the if-condition ( c == alphabets[i] ): it is comparing each alphabet with

console.log(msg)
```

```
C IS AN ALPHABET
```

```
let c = '+'; // Simple variable declaration. Variable of type: “string”

let alphabets = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', '
// Another simple variable declaration. Variable of type: Array

let msg = 'C IS NOT AN ALPHABET'
// ‘msg’ is of type: String

for (let i = 0; i < alphabets.length; i++) {
    if (c == alphabets[i]){
        msg = "C IS AN ALPHABET"
    }
}

// To iterate over ‘alphabets’ we need a variable called ‘i’.
// i starts taking values from 0 till alphabets.length (which is 26)
// Next in the if-condition ( c == alphabets[i] ): it is comparing each alphabet with the contents of v

console.log(msg)
```

```
C IS NOT AN ALPHABET
```

Two Approaches to Solve This Problem

Approach 1:

We have seen that we can compare contents of variable ‘c’ with all the alphabets to see if we get a match.

This is an iterative solution.

Approach 2:
We search for alphabets in the contents of ‘c’.

This would leverage RegEx.

How can we capture an alphabetical character using RegEx?

```
let d = '5'

d.search('a')
-1

d.search('5')
0

d.search(/[a-z]/)
-1

d = 'f'
"f"

d.search(/[a-z]/)
0

d = '5'
"5"

d.search(/[0-9]/)
0
```

Detecting digits

With regular expressions, you don’t need to know what exactly you want to find.

What you should know is a rough idea about what that you are trying to find looks like.

Q: I want to find if a digit is present in a string.
A:
[] Represents a class of characters.

[0-9] Represents a class of the 10 digits.

You can find a digit by iterating over range(0, 10) and find it a digit is present in a string.
Or, you can tell the computer that if you find any digit from 0 to 9, then let me know.

Remember the constituent components to form a RegEx

Forming a Regex for a set of strings requires the developer to remember the building blocks of a Regex pattern.

It is like remembering the number tables to do multiplication fast.

Even if you don’t remember the building blocks of a Regex pattern, you can always refer to the internet.

Modifiers

Modifier	Description
g	Perform a global match (find all matches rather than stopping after the first match)
i	Perform case-insensitive matching
m	Perform multiline matching

/pattern/modifier(s);

let pattern = /w3schools/i;

Brackets

Expression Description

[abc] Find any character between the brackets.
Here, example pattern says ‘let me know if a or b or c appears in the text.’

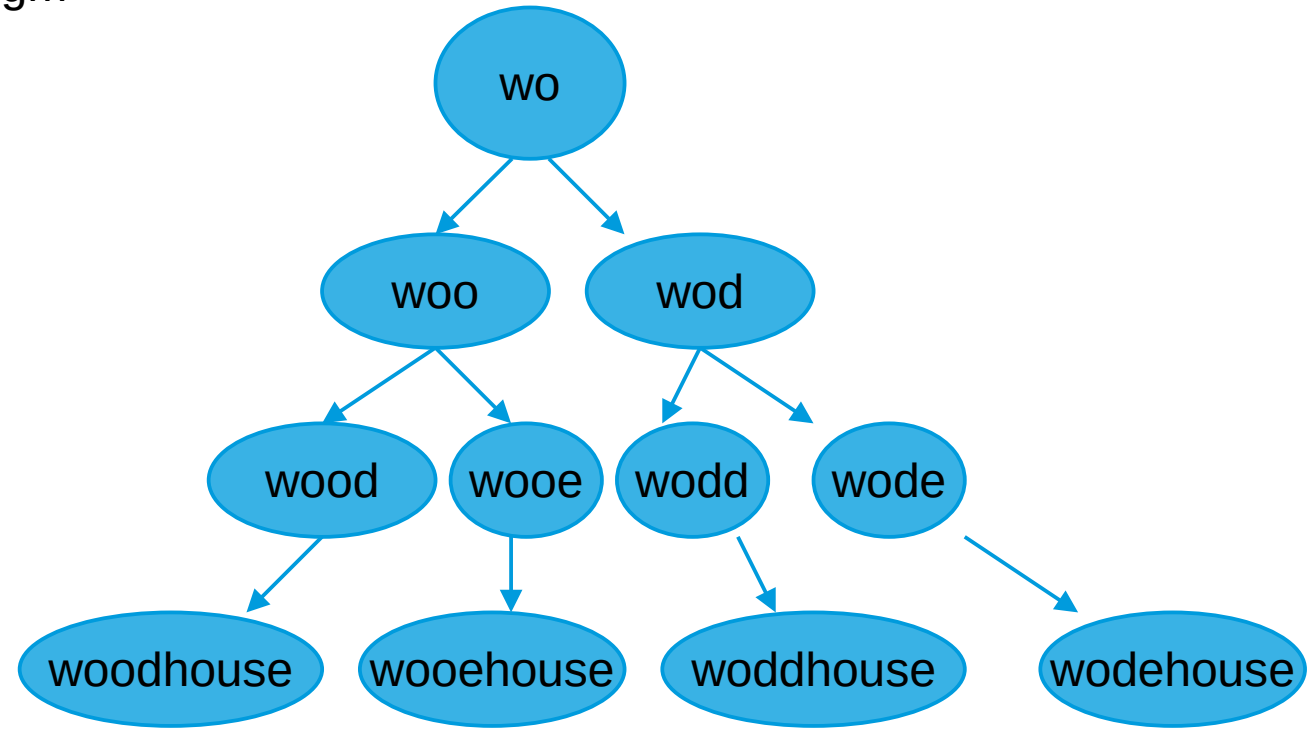
[^abc] Find any character NOT between the brackets.
Here, example pattern says ‘let me if a character other than a, b and c appears in the text’

[0-9] Find any character between the brackets (any digit)
[^0-9] Find any character NOT between the brackets (any non-digit)

(x|y) Find any of the alternatives specified
For example: /woodhouse|woodcock/gm
Here, example pattern says ‘let me know if either if you either find woodhouse or woodcock in the text and modifier g: means globally, m: means multiline search’

Square Brackets

What all can this pattern match?
/wo[od][de]house/gm



Metacharacters

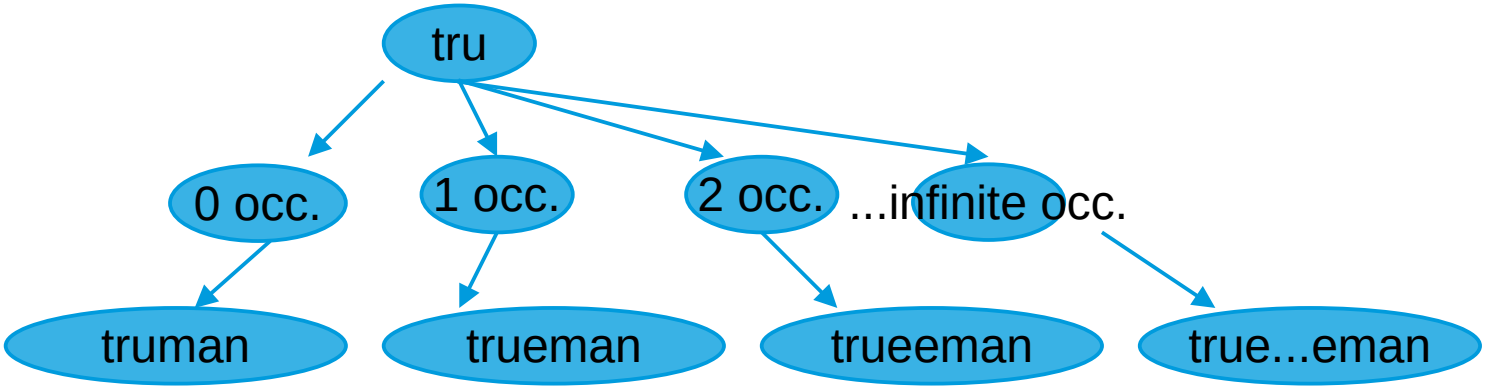
Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word, beginning like this: \bHI, end like this: HI\b
\B	Find a match, but not at the beginning/end of a word
\0	Find a NULL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\udddd	Find the Unicode character specified by a hexadecimal number dddd

Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{X}	Matches any string that contains a sequence of X n's
n{X,Y}	Matches any string that contains a sequence of X to Y n's
n{X,}	Matches any string that contains a sequence of at least X n's
n$	Matches any string with n at the end of it
^n	Matches any string with n at the beginning of it
?=n	Matches any string that is followed by a specific string n
?!n	Matches any string that is not followed by a specific string n

/true*man/gm

n* Matches any string that contains zero or more occurrences of n



/p.*woodhouse/gm

. Find a single character, except newline or line terminator (It is a metacharacter)

. means ‘any character’
.* means ‘any character any number of times’ (0 or more)

p.*woodhouse: String would start from ‘p’ then ‘any character any number of times’. Then string would end in ‘woodhouse’.

Let’s try out ‘p.*woodhouse’ for following strings:
p.woodhouse
p.j. woodhouse
p.k. woodhouse

```
let pattern = /p.*woodhouse/
let s = 'p.woodhouse'
pattern.test(s)
true
```

```
pattern = /p.*woodhouse/
s = 'p.k.woodhouse'
pattern.test(s)
true
```

```
pattern = /p.*woodhouse/
s = 'p.k.woodhouse'
pattern.test(s)
true
```

^n, [^n], ^[^n] : what's the difference?

^n
This caret is anchoring the pattern. It says return the string if it starts with 'n'
For ex: /^2/ means 'if string starts with 2'.

[^n]
This caret is negating the class. It says "Find any character NOT between the brackets".

^[^n]
The two carets here have totally different meanings.
The first one anchors the pattern.
And the other negates a class.
For ex: /^[^2]/ means 'if string starts any character other than 2'.

[78]....$

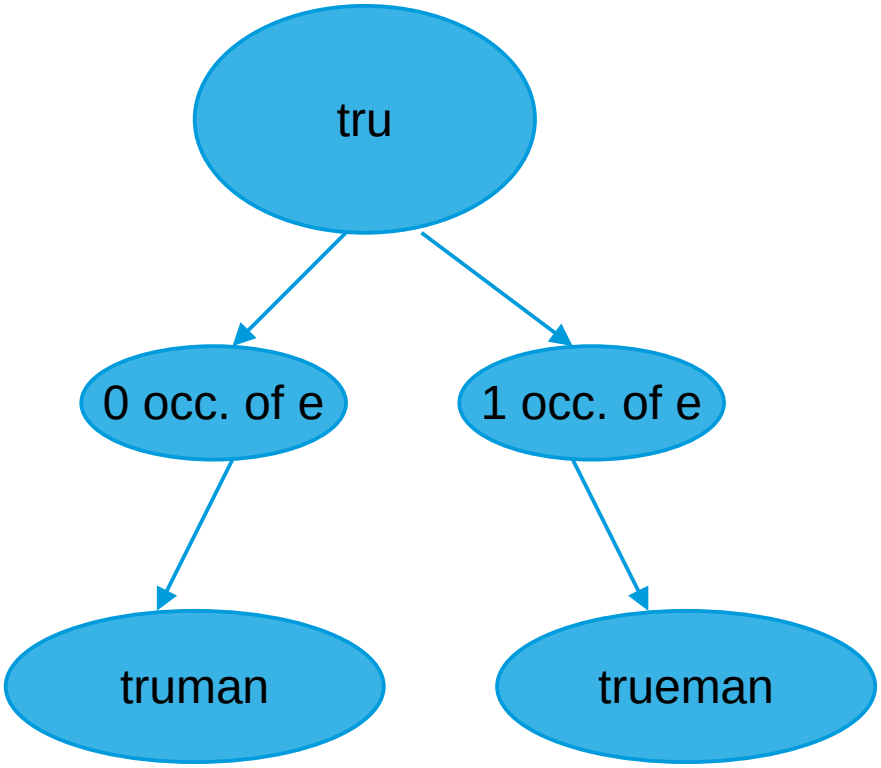
Quantifier $
n$ Matches any string with n at the end of it

[78] means either of 7 or 8
. This metacharacter means ‘any character’

To simplify the case to be able to explain, if we assume that the string we are looking for is a number.
Then /[78]..../ would represent any number between 70000 to 89999

$ simply means that the string would appear at the end of the input text

/true?man/gm



Problem ‘zoo’

You are required to enter a word that consists of x and y that denote the number of Zs and Os respectively. The input word is considered similar to word zoo if

2 * x = y

Determine if the entered word is similar to word zoo.

For example, words such as zzoooo and zzzooooooo are similar to word zoo but not the words such as zzooo and zzzooooo.

Input format

First line: A word that starts with several Zs and continues by several Os.

Note: The maximum length of this word must be 20

Output format

Print Yes if the input word can be considered as the string zoo otherwise, print No.

Solution

```
/* Counting z’s and o’s using iteration */
x = 'zoo'
z_count = 0;

for (i = 0; i < x.length; i++) {
    if (x[i] == 'z'){
        z_count += 1;
    }
}
console.log(z_count)

o_count = 0;
for (i = 0; i < x.length; i++) {
    if (x[i] == 'o'){
        o_count += 1;
    }
}
console.log(o_count)
```

Count z's and o's using regex

```
x = 'zoo'

let myArr = x.match(/z/g);
z_count = myArr.length
console.log(z_count);

myArr = x.match(/o/g);
o_count = myArr.length;
console.log(o_count);
```

Check that z's come before o's

```
x = 'zoos'
pattern = /^[z]+[o]+$/g;
console.log(pattern.test(x));

/*
Test cases:

1.
zoo : true

2.
zoos : false

*/
```

Node.js Based Solution

```
// Sample code to perform I/O:
console.log("Reading input started, press Ctrl+D to submit and close input process.");
process.stdin.resume();
process.stdin.setEncoding("utf-8");
var stdin_input = "";
process.stdin.on("data", function (input) {
    stdin_input += input;                           // Reading input from STDIN
});
process.stdin.on("end", function () {
   main(stdin_input);
});
function main(input) {
    console.log("\n");

    x = input
    let myArr = x.match(/z/g);
    z_count = myArr.length
    myArr = x.match(/o/g);
    o_count = myArr.length;
    if (o_count != 2 * z_count) {
        console.log("No, number of Os is not double of number of Zs.");
        return
    }
    pattern = /^[z]+[o]+$/g;
    if(pattern.test(x)) {
        console.log('Yes, both tests cleared.');
    } else {
        console.log('No. Input is not following the pattern ^[z]+[o]+$');
    }
}
```