//ABAIXO SEGUE UMA ESPÉCIE DE DICIONÁRIO DO REGEX:

//OBS: Ele começa com sinais de pontuação, números e depois letras

/\*

    .   =   O ponto "." é um meta-char universal que pode significar qualquer caractere.

    \   =   O escape "\" é usado para quando desejamos usar um meta-char especial ou um sinal comum sem ativar um

            meta-char;

    ?   =   A interrrogação "?" é um símbolo para apresentar um valor que pode ocorrer zero ou uma vez, ou seja, ele dá a

                opção: se tiver o caractere uma vez, retorne ele, mas se não tiver, não tem problema, pode retornar sem

                esse caractere também. Ele sempre é usado em conjunto com uma classe ou meta-char. A interrrogação é um

                dos quantifier do ReGex, ele significa zero ou uma vez;

    \*   =   O asteristico "\*" é um meta-char quantifier também, ele significa zero ou muitos, usamos ele em conjunto com

                classes e meta-chars para determinar que um caractere pode ocorrer zero ou muitas vezes;

    +   =   O sinal de "+" é um meta-char quantifier também, ele significa uma ou muitas vezes, usamos ele em conjunto

                com classes e meta-chars para determinar que um caractere possa ocorrer uma ou muitas vezes;

    ^   =   O circunflexo "^" é um meta-char para determinar uma negação, geralmente o usamos precedido a um carectere

                ou caracteres que desejamos que não sejam encontrados;

    $   =   O Sifrão "$" é um meta-char âncora para demilitar um fim da linha, usamos para encontrar palavras que

                precedem um fim da linha, dessa forma "abc$", toda palavra "abc" que estiver no fim de uma linha será

                encontrada;

    {}  =   As chaves "{}" são usadas para quando desejamos usar uma determinada quantidade de meta-chars, inclusive as

            chaves são um dos "quantifier" usados no ReGex. As chaves aceitam os seguintes valores:

                {n} significa "n" vezes determinadas que um caractere pode ocorrer;

                {n,} significa que um caractere pode ocorrer no mínimo "n" vezes;

                {n,m} significa que um caractere pode ocorrer no mínimo "n" vezes e no máximo "m" vezes;

    []  =   Os colchetes "[]" delimitam uma classe de caracteres, classes de caracteres são um conjunto de caracteres

                que desejamos usar encontrar um determinado conjunto de caracteres. Podemos delimitá-los pela "," ou "-", usamos "," quando queremos um ou outro caractere, e usamos "-" quando desejamos ter valores de 1 determinado caractere até outro determinado caractere. Segue abaixo alguns exemplos de classes:

                        [a-z] significa que podemos encontrar qualquer letra minúscula que não seja acentuada;

                        [A-Z] significa que podemos encontrar qualquer letra maiúscula que não seja acentuada;

                        [A-z] significa que podemos encontrar qualquer letra maiúscula ou minúscula não acentuada;

                        [0-9] significa que podemos encontrar qualquer algarismo entre 0 e 9;

                        [n,m] significa que podemos encontrar "n" ou "m" caractere (pode ser usado qualquer valor);

                        [0-9A-zç] veja que podemos mesclar para encontrar caracteres dos mais variados;

                Atenção, lembre-se que dentro de uma classe somente os caracteres "\" e "^" são meta-chars;

    \b  =   O escape b "\b" é uma âncora também, significa boundary (fronteira), para encontrar uma expressão que não

                pode ser antecedida, precedida ou ambos por nenhum char que não esteja entre o "\b", geralmente a usamos

                assim: "\bexpressao\b" ela vi procurar somente as ocorrêcias onde a palavra ocorre;

    \B  =   O escape B "\B" (maiúsculo) é uma âncora que faz o contrário do "\b", por isso a chamamos e

                Non-Word-Boundary, ou seja, uma expressão regex que só poderá estar no meio de uma palavra, ela nunca

                poderá ser uma palavra fronteiriça.

    \d  =   O espape d "\d" é um meta-char para encontrar dígitos;

    \s  =   O escape s "\s" é um meta-char para encontrar espaços entre os caracteres;

    \w  =   O escape w "\w" é um meta-char para encontrar qualquer caractere alfanumérico e o underline (ATENÇÃO: ele

                não encontra caracteres acentuados ou especiais);

\*/

//TARGET USADO EM TODOS OS EXEMPLO:

let targetDigitos = "João Almeida Augusto de Melo - Professor - Idade 36 - CPF 349.832.441-23 - RG 34.435.534-2 CNPJ 15.123.321/8883-22"

//USANDO PONTO:

let ExPonto = RegExp(/..\d/) //Perceba que o ponto vai pegar quaisquer 2 primeiros caracteres que estiverem antes do 1º

                                //dígito que for encontrado...

console.log("Exemplo de Ponto: " + ExPonto.exec(targetDigitos)) //Usamos aqui a função "exec" do RegExp essa função

                                                                    //retorna um array, onde o índice 0 é o resultado

                                                                    //que procuramos, o índice 1 traz o valor índice do

                                                                    //resultado que procuramos, o valor 2 traz o target

                                                                    //completo. Mas quando usamos o exec concatenado com

                                                                    //uma string ele traz automáticamente o índice 0 da

                                                                    //função "exec". Além da função exec temos muitas

                                                                    //outras, para mais informações consulte a

                                                                    //especificação oficial no site

                                                                    //https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Regular\_Expressions

//USANDO DÍGITOS: (Aqui a missão é encontrar o CPF)

let ExDigitos = RegExp(/\d\d\d\.\d\d\d\.\d\d\d-\d\d/) //Perceba que podemos colocar um "\d" um atrás do outro...

let ExDigitos2 = RegExp(/\d{3}\.\d{3}\.\d{3}-\d{2}/) //Mas uma solução mais elegante seria usar o quantifier para

                                                        //delimitar a quantidade de dígitos. Perceba também que estamos

                                                        //usando o "\." para referenciar as pontuações do CPF

console.log("Exemplos de Dígitos: " + ExDigitos.exec(targetDigitos)) //Perceba que o resultado será o mesmo...

console.log("Exemplos de Dígitos: " + ExDigitos2.exec(targetDigitos))

//USANDO ESCAPE PARA PEGAR CARACTERES ESPECIAIS:

let ExBarra = RegExp(/\d{2}\.\d{3}.\d{3}\/\d{4}-\d{2}/) //Quando queremos a barra, temos que usar o escape junto com a

                                                            //barra, dessa forma "\/" para que consigamos referenciar a

                                                            //barra...

console.log("Exemplos de Barra: " + ExBarra.exec(targetDigitos)) //Aqui queremos como resultado o CNPJ da pessoa...

//USANDO CLASSES DE CARACTERE:

//EXEMPLO COM VÍRGULA:

let ExClasseVirgula = RegExp(/[0,1,2,3][0,1,2,3][0,1,2,3]/)//Perceba que dentro das classes estamos usando números

                                                                //separados por vírgula de 0 á 3, cada classe vai representar apenas 1 dígito, ou seja, estamos pesquisando pelo conjunto de 3 dígitos onde cada dígito deverá estar entre 0 e 3....

console.log("Exemplo de Classe com separação por vírgula para para os valores 0 ou 3: " + ExClasseVirgula.exec(targetDigitos)) //veja que pegamos o segundo conjunto de dígitos do CNPJ...

//EXEMPLO COM TRAÇO:

let ExClasseTraco = RegExp(/[0-9][0-3][0-2]/)//Perceba que dentro das classes estamos usando números separados por

                                                //traço tentando pegar um conjunto de 3 dígitos onde o 1º dígito poderá conter valores que vão de 0 á 9, o 2º poderá conter valores de 0 á 3 e o 3º poderá ir de 0 á 2...

console.log("Exemplo de Classe com separação por vírgula para para os valores 0 ou 3: " + ExClasseTraco.exec(targetDigitos)) //veja que pegamos o segundo conjunto de dígitos do CPF...

//USANDO ? PARA PEGAR 0 OU ALGUM VALOR:

let ExSemInterrogacao = RegExp(/[a-z][a-z]/) //Nosso objetivo aqui é pegar o 1º conjunto de 2 letras minúsculas sem

                                                //acentuação.

let ExInterrogacao = RegExp(/[a-z][a-z]?/) //Mas aqui nós já damos a opção de que pode ser 1 letra minúscula sem

                                                //acentuação, mas se tiver 2 pode trazer também.

console.log("Exemplo com Interrogação: " + ExSemInterrogacao.exec(targetDigitos)) //Veja que quando queremos 2 letras ele

                                                                                        //força o encontro do primeiro conjunto e letras minúsculas sem acentuação, indo até o "lm" do Almeida..

console.log("Exemplo com Interrogação: " + ExInterrogacao.exec(targetDigitos)) //Mas quando tornamos o encontro da 2

                                                                                //letra opcional, ele pega para nós o 1º caractere "o" do nome João...

//USANDO QUANTIFERS PARA ENCONTRAR UMA DATA PADRÃO:

let data = "01/02/21" //Note que para encontrar datas temos que seguir um determinado padrão, onde os 2 primeiros

                        //dígitos só poderão ir do 01 ao 31, os 2 segundos dígitos só poderão ir do 01 ao 12 e o ano

                        //poderá ter 2 ou 4 dígitos...

let data2 = "31/12/2021"

let ExData = new RegExp(/[0-3]?\d\/[0-1]?\d\/\d{2,4}/)

console.log("Exemplo com Quantifiers: " + ExData.exec(data)) //Note que o ReGex funciona para ambas as datas...

console.log("Exemplo com Quantifiers: " + ExData.exec(data2))

//USANDO ESPAÇOS COM \s:

let ExEspacos = new RegExp(/[A-zã]+\s[A-z]+\s[A-z]+/) //Veja que usamos as classes alfabéticas para encontrar qualquer

                                                        //sequência de letras maiúsculas ou minúsculas contendo 1 ou mais

                                                        //ocorrências, seguido pelo \s, ou seja, a cada espaço ela passa

                                                        //para a próxima sequência. Note também que na primeira classe

                                                        //alfabética usamos em conjunto o "ã" para pegar o acentuação

                                                        //que existe no nome "João"

console.log("Exemplo com Espaço \\s: " + ExEspacos.exec(targetDigitos))

//USANDO O CIRCUNFLEXO PARA IGNORAR CARACTERES:

let ExCircunflexo = new RegExp(/[^0-9]\*/) //Note que estamos usando a negação para não pegar nenhum algarismo, somente as

                                                //letras, mas note que quando ele encontra o primeiro algarismo o regex

                                                //encerra a comparação. A negação é muito boa para utilizar quando

console.log("Exemplo de negação com circunflexo: " + ExCircunflexo.exec(targetDigitos))

//USANDO O \w PARA ENCONTRAR QUALQUER CARACTERE ALFANUMÉRICO:

let ExLetras = new RegExp(/[\wã\s-\.]+/) //Note que estamos usando o "\w" para pegar todos os caracteres alfanuméricos

                                                //e tivemos que acrescentar alguns caracteres qua não seriam encontrados

                                                //pelo "\w"....

console.log("Exemplo de Letras com \\w: " + ExLetras.exec(targetDigitos))

//USANDO $ PARA ENCONTRAR PALAVRAS QUE MARCAM O FIM DE UMA LINHA:

let ExSifrao = new RegExp(/\w$/) //Perceba que usamos o "$" logo após qualquer caractere alfanumérico que esteja no final

                                        //da linha, que no caso é o caracter "2" do CNPJ

console.log("Exemplo de Sifrão: " + ExSifrao.exec(targetDigitos))

//USANDO \b COMO ANCORA PARA ENCONTRAR SOMENTE UMA DETERMINADA EXPRESSÃO:

let ExAncoraB = new RegExp(/\bde\b/) //Nesse caso queremos encontrar uma expressão "de" que esteja isolada no texto, ela

                                        //não pode estar sendo precedida por nenhum caractere e nem ser precedida por

                                        //nenhum outro caractere, só poderá ter espaços entre ela, como resultados temos

                                        //o "de" do "Augusto de Melo"...

console.log("Exemplo de Âncora \\b: " + ExAncoraB.exec(targetDigitos))

//USANDO \B COMO ANCORA PARA ENCONTRAR SOMENTE UMA EXPRESSÃO QUE ESTIVER NO MEIO DE UMA PALAVRA:

let ExAncoraBMaiusculo = new RegExp(/.+\Bde\b/) //Nesse caso queremos encontrar uma expressão "de" que esteja no meio de

                                                        //uma expressão, mas que não pode ser precedida por nenhuma outra

                                                        //letra...

console.log("Exemplo de Âncora \\B: " + ExAncoraBMaiusculo.exec(targetDigitos))

RESULTADOS NO CONSOLE...

[Running] node "c:\Users\Almoxarifado\Documents\Gah\javascript\arquivos\_das\_aulas\199-REGEX\_\_Dicionario\_do\_ReGeX.js"

Exemplo de Ponto: e 3

Exemplos de Dígitos: 349.832.441-23

Exemplos de Dígitos: 349.832.441-23

Exemplos de Barra: 15.123.321/8883-22

Exemplo de Classe com separação por vírgula para para os valores 0 ou 3: 123

Exemplo de Classe com separação por vírgula para para os valores 0 ou 3: 832

Exemplo com Interrogação: lm

Exemplo com Interrogação: o

Exemplo com Quantifiers: 01/02/21

Exemplo com Quantifiers: 31/12/2021

Exemplo com Espaço \s: João Almeida Augusto

Exemplo de negação com circunflexo: João Almeida Augusto de Melo - Professor - Idade

Exemplo de Letras com \w: João Almeida Augusto de Melo - Professor - Idade 36 - CPF 349.832.441-23 - RG 34.435.534-2 CNPJ 15.123.321

Exemplo de Sifrão: 2

Exemplo de Âncora \b: de

Exemplo de Âncora \B: João Almeida Augusto de Melo - Professor - Idade

[Done] exited with code=0 in 0.177 seconds