

[Home](#) → [Regular expressions](#)

📅 19th December 2019

Unicode: flag "u" and class \p{...}

JavaScript uses [Unicode encoding](#) for strings. Most characters are encoding with 2 bytes, but that allows to represent at most 65536 characters.

That range is not big enough to encode all possible characters, that's why some rare characters are encoded with 4 bytes, for instance like \mathcal{X} (mathematical X) or 😊 (a smile), some hieroglyphs and so on.

Here are the unicode values of some characters:

Character	Unicode	Bytes count in unicode
a	0x0061	2
≈	0x2248	2
\mathcal{X}	0x1d4b3	4
\mathcal{Y}	0x1d4b4	4
😊	0x1f604	4

So characters like a and ≈ occupy 2 bytes, while codes for \mathcal{X} , \mathcal{Y} and 😊 are longer, they have 4 bytes.

Long time ago, when JavaScript language was created, Unicode encoding was simpler: there were no 4-byte characters. So, some language features still handle them incorrectly.

For instance, `length` thinks that here are two characters:

```
1 alert('😊'.length); // 2
2 alert('X'.length); // 2
```



...But we can see that there's only one, right? The point is that `length` treats 4 bytes as two 2-byte characters. That's incorrect, because they must be considered only together (so-called "surrogate pair", you can read about them in the article [Strings](#)).

By default, regular expressions also treat 4-byte "long characters" as a pair of 2-byte ones. And, as it happens with strings, that may lead to odd results. We'll see that a bit later, in the article [Sets and ranges \[...\]](#).

Unlike strings, regular expressions have flag u that fixes such problems. With such flag, a regexp handles 4-byte characters correctly. And also Unicode property search becomes available, we'll get to it next.

Unicode properties \p{...}

⚠ Not supported in Firefox and Edge

Despite being a part of the standard since 2018, unicode properties are not supported in Firefox ([bug](#)) and Edge ([bug](#)).

There's [XRegExp](#) library that provides “extended” regular expressions with cross-browser support for unicode properties.

Every character in Unicode has a lot of properties. They describe what “category” the character belongs to, contain miscellaneous information about it.

For instance, if a character has `Letter` property, it means that the character belongs to an alphabet (of any language). And `Number` property means that it's a digit: maybe Arabic or Chinese, and so on.

We can search for characters with a property, written as `\p{...}`. To use `\p{...}`, a regular expression must have flag `u`.

For instance, `\p{Letter}` denotes a letter in any of language. We can also use `\p{L}`, as `L` is an alias of `Letter`. There are shorter aliases for almost every property.

In the example below three kinds of letters will be found: English, Georgian and Korean.

```
1 let str = "A ზ ა";
2
3 alert( str.match(/\p{L}/gu) ); // A,ზ,ა
4 alert( str.match(/\p{L}/g) ); // null (no matches, as there's no flag "u")
```

Here's the main character categories and their subcategories:

- Letter `L` :
 - lowercase `Ll`
 - modifier `Lm`,
 - titlecase `Lt`,
 - uppercase `Lu`,
 - other `Lo`.
- Number `N` :
 - decimal digit `Nd`,
 - letter number `Nl`,
 - other `No`.
- Punctuation `P` :
 - connector `Pc`,
 - dash `Pd`,
 - initial quote `Pi`,
 - final quote `Pf`,
 - open `Ps`,
 - close `Pe`,
 - other `Po`.
- Mark `M` (accents etc):
 - spacing combining `Mc`,

- enclosing Me ,
- non-spacing Mn .
- Symbol S :
 - currency Sc ,
 - modifier Sk ,
 - math Sm ,
 - other So .
- Separator Z :
 - line Zl ,
 - paragraph Zp ,
 - space Zs .
- Other C :
 - control Cc ,
 - format Cf ,
 - not assigned Cn , – private use Co ,
 - surrogate Cs .

So, e.g. if we need letters in lower case, we can write \p{Ll} , punctuation signs: \p{P} and so on.

There are also other derived categories, like:

- Alphabetic (Alpha), includes Letters L , plus letter numbers Nl (e.g. XII – a character for the roman number 12), plus some other symbols Other_Alphabetic (OAlpha).
- Hex_Digit includes hexadecimal digits: 0-9 , a-f .
- ...And so on.

Unicode supports many different properties, their full list would require a lot of space, so here are the references:

- List all properties by a character: <https://unicode.org/cldr/utility/character.jsp>.
- List all characters by a property: <https://unicode.org/cldr/utility/list-unicodeset.jsp>.
- Short aliases for properties: <https://www.unicode.org/Public/UCD/latest/ucd/PropertyValueAliases.txt>.
- A full base of Unicode characters in text format, with all properties, is here: <https://www.unicode.org/Public/UCD/latest/ucd/>.

Example: hexadecimal numbers

For instance, let's look for hexadecimal numbers, written as xFF , where F is a hex digit (0...1 or A...F).

A hex digit can be denoted as \p{Hex_Digit} :

```
1 let regexp = /x\u{Hex_Digit}\u{Hex_Digit}/u;
2
3 alert("number: xAF".match(regexp)); // xAF
```



Example: Chinese hieroglyphs

Let's look for Chinese hieroglyphs.

There's a unicode property `Script` (a writing system), that may have a value: `Cyrillic` , `Greek` , `Arabic` , `Han` (Chinese) and so on, [here's the full list](#).

To look for characters in a given writing system we should use `Script=<value>` , e.g. for Cyrillic letters: `\p{sc=Cyrillic}` , for Chinese hieroglyphs: `\p{sc=Han}` , and so on:

```
1 let regexp = /\p{sc=Han}/gu; // returns Chinese hieroglyphs
2
3 let str = `Hello Привет 你好 123_456`;
4
5 alert( str.match(regexp) ); // 你,好
```

Example: currency

Characters that denote a currency, such as `$` , `€` , `¥` , have unicode property `\p{Currency_Symbol}` , the short alias: `\p{Sc}` .

Let's use it to look for prices in the format "currency, followed by a digit":

```
1 let regexp = /\p{Sc}\d/gu;
2
3 let str = `Prices: $2, €1, ¥9`;
4
5 alert( str.match(regexp) ); // $2,€1,¥9
```

Later, in the article [Quantifiers +, *, ? and {n}](#) we'll see how to look for numbers that contain many digits.

Summary

Flag `u` enables the support of Unicode in regular expressions.

That means two things:

1. Characters of 4 bytes are handled correctly: as a single character, not two 2-byte characters.
2. Unicode properties can be used in the search: `\p{...}` .

With Unicode properties we can look for words in given languages, special characters (quotes, currencies) and so on.

[Previous lesson](#)[Next lesson](#)

Share  

 [Tutorial map](#)

Comments

- If you have suggestions what to improve - please [submit a GitHub issue](#) or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>` , for more than 10 lines – use a sandbox ([plnkr](#), [JSBin](#), [codepen](#)...)