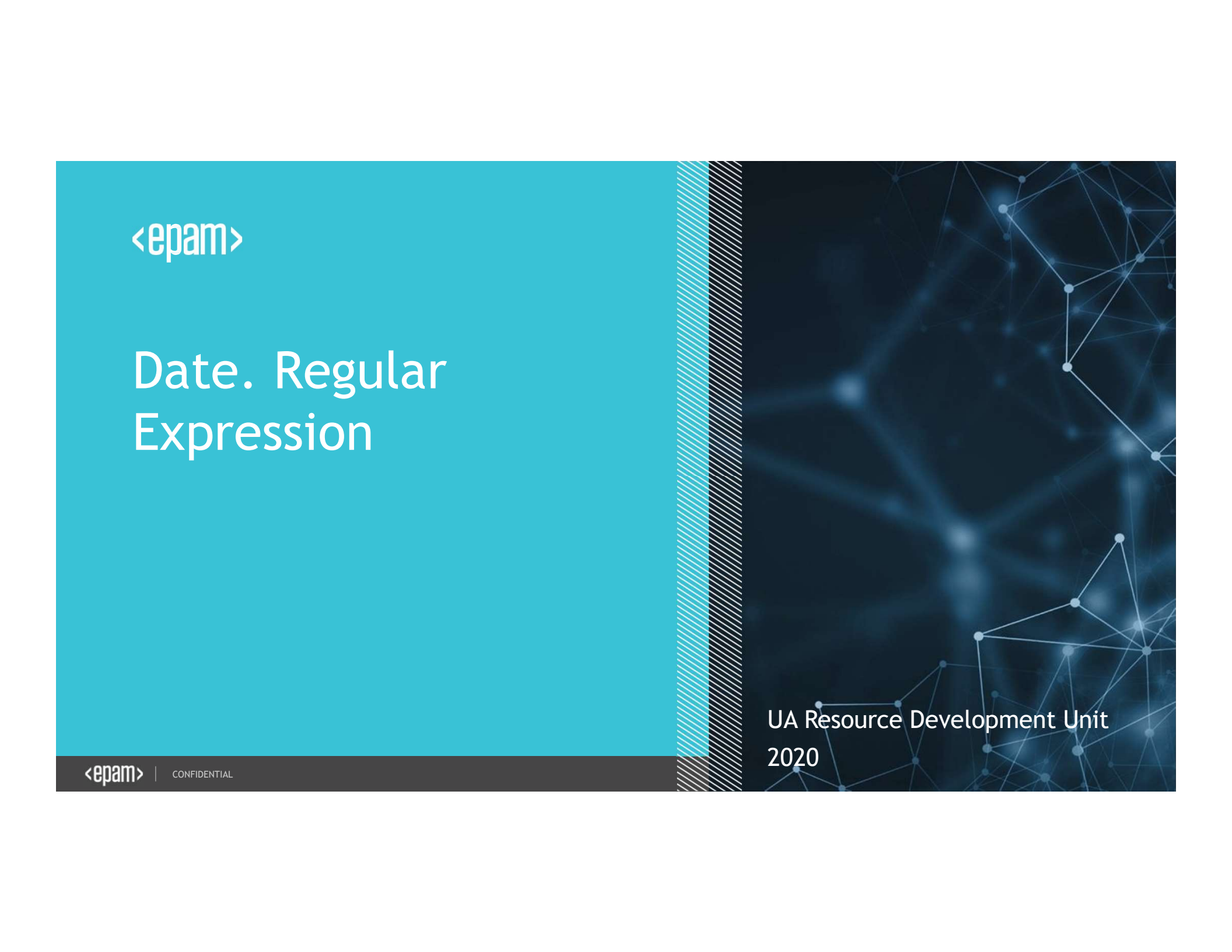
Date. Regular Expression
UA Resource Development Unit
2020
CONFIDENTIAL

AGENDA

1. Date
2. Date Syntax
3. Data get/set methods
4. Date transform methods
5. Work with Regular Expressions

DATE

Date constructor

Create date:

```
new Date();
new Date(value);
new Date(dateString);
new Date(year, month[, day[, hour[, minute[, second[, millisecond]]]]]);
```

Examples:

```
new Date() // "Thu Feb 28 2019 14:54:18 GMT+0200 (Eastern European Standard Time)"
new Date(1463295600000);
new Date('May 15, 2016 10:00:00');
new Date(2016, 4, 15, 10);
```

Date get methods

```
Date.prototype.getDay(); // returns the day of the week (0 means Sunday)
Date.prototype.getDate(); // returns the day of the month
Date.prototype.getFullYear(); // returns the year (Y2K compliant)
Date.prototype.getHours();
Date.prototype.getMilliseconds();
Date.prototype.getMinutes();
Date.prototype.getMonth(); // returns the month (0 means January)
Date.prototype.getSeconds();
Date.prototype.getTime(); // returns the milliseconds since UNIX Epoch
Date.prototype.getTimezoneOffset(); // time zone difference in minutes (from curent
local to UTC)
```

Date set methods

```
Date.prototype.setFullYear(year [, month, date]);
Date.prototype.setMonth(month [, date]);
Date.prototype.setDate(date); // set the day of the month
Date.prototype.setHours(hour [, min, sec, ms]);
Date.prototype.setMinutes(min [, sec, ms]);
Date.prototype.setSeconds(sec [, ms]);
Date.prototype.setMilliseconds(ms);
Date.prototype.setTime(milliseconds); // milliseconds since UNIX Epoch
```

Date transform methods

```
Date.prototype.toDateString(); // example output: "Wed Jul 28 1993 14:39:07 GMT+0200 (CEST)"
Date.prototype.toISOString(); // example: "2021-02-24T15:18:23.691Z"
Date.prototype.toJSON();
Date.prototype.toLocaleString(); // example: "2021. 02. 24. 15:23:47"
Date.prototype.toLocaleDateString();
Date.prototype.toLocaleTimeString();
Date.prototype.toUTCString();
```

REGULAR EXPRESSION

RegExp

Regular expression is a sequence of characters that define a search pattern. Usually this pattern is used for “find” or “find and replace” operations on strings.

We can create regular expression in two ways:

EXAMPLE

```
// 1) Without constructor
const simpleRegExp = /pattern/;
// 2) With constructor
const anotherRegExp = new RegExp('pattern');
```

Note: Don’t modify the RegExp object!

It will make the work of a regular expression much slower.

RegExp: flags

There are such flags as:

- g — Global match
- i — Ignore case
- m — Multiline (^ and $ work for each line)
- ... — etc

EXAMPLE

```
'this\nawesome end\nisn\'t\nend'.match(/.*end$/gm);
// ["awesome end", "end"]

/case/i.test('cAsE'); // true
```

RegExp: special characters

There are a lot of special characters for regular expressions, it's quite hard to remember all of them, here are some examples:

- \d and \D – any digit(d) and non-digit(D)
- \s and \S – any space(s) and non-space(S) character
- […] and [^…] – any character from brackets and any character not from brackets (with ^)
- [a-z] – any character in range from "a" to "z"
- a* – zero or many matches of the character "a"
- a+ – one or more matches of the character "a"
- a? – zero or one match of the character "a"
- a{2,3} – 2 or 3 matches of "a", also can be a{2}(exactly 2) or a{2,}(2 or more)
- ^ - beginning of the string
- $ - end of the string
- And so on…

RegExp: groups

We can capture something specific from the string using groups.
To declare a group you have to put the part of your regular expression into round brackets.

There are special types of groups, such as: capturing/non-capturing, named, following/followed.

Also if we need to match one of several cases we can use special separator: /(a|b)/ - matches “a” or “b”

For example it can be used for taking a specific value (*user_token*) from cookies:

EXAMPLE

```
const cookie = document.cookie;
// user_token=12345abc11; numeric_key=88; type=rsa;

const match = /(?:^|; )user_token=([^;]*)/.exec(cookie);
const userToken = match ? match[1] : undefined;
// userToken === '12345abc11';
```

Regular Expressions

WORK WITH EXPRESSIONS

RegExp: RegExp-object methods

RegExp.prototype.test() accepts a string as an argument, returns a Boolean value that indicates if there is a match in the string for the pattern.

RegExp.prototype.exec() accepts a string as an argument, returns a result array or null if there are no matchings for the pattern.

EXAMPLE

```
/\d$/.test('I have a number - 5'); // true
/^\d/.test('I have a number - 5'); // false

const text = 'You can find here animals like dogs, cats and
so on.';

const result = /like.+(cats).+(sheep)?.+so\son/.exec(text);
/*
result = [
    "like dogs, cats and so on",
    "cats",
    undefined,
    index: 26,
    input: "You can find here animals like dogs, cats and
so on."
];
*/
```

RegExp: string methods

String.prototype.match() is a method that returns an array of all matches in a string for passed regexp, or null.

String.prototype.search() is a method that returns the index of the matched part of string or -1 if there is no match.

String.prototype.split() can accept a RegExp as an argument to split a string to an array.

String.prototype.replace() is a method that searches for match in a string and replaces it with a given substring.

EXAMPLE

```
'Hello world'.match(/^\w[^\s]+/);
// ["Hello"]

'There is a color #a0a here'
    .search(/#(?:[a-f0-9]{3}){1,2}/);
// 17

'Apple, Orange,Pear,   Cherry'
    .split(/,\s*/);
// ["Apple", "Orange", "Pear", "Cherry"]

'Hello {name} {brackets}!'
    .replace(/{.+}/, 'Vasya'); // ?
```

RegExp

A regular expression is an object that describes a pattern of characters. Regular expressions are used to perform pattern-matching and "search-and-replace" functions on text.

```
/word/ pattern that matches "word", using regex syntax
var regexp = new RegExp('word', ''); pattern that matches "word"
using RegExp object
. matches any character (except newline)
/o.o/ matches 'oao', 'obo', 'o o', 'o-o' ...
^ indicates beginning of line
$ indicates end of line
/^word$/ matches lines that consist only of 'word'.
```

RegExps are matched **line by line**.
http://regexr.com/
http://regexper.com/

RegExp Methods

RegExp.test(string); tests "string" to match regexp.

```
/world/.test('hello, world!'); // -> true
```

String.match(regexp); matches string, returning null if no matches found, array of matches otherwise.

```
'hello, world'.match(/world/); // -> ["world"]
```

Also regexp may be used in ''.replace:

```
'testtest'.replace(/test/, function (match) {
    return match + '.';
}); // test.test
```

RegExp Grouping

Groups:

- [] - Character set, match any of given characters
- [a-z] - Any of lowercase letters.
- (...) - create "match group". Groups are accessible as $1..$9
- 'hello world'.replace(/(\w+)/g, '-$1-'); // "-hello- -world-"

RegExp Repeating

{count} - previous symbol repeated exactly 'count' times
Example: /a{2}/ - matches 'aa’

{x, y} - previous symbol (or group) repeated from "x" to "y" times
Example: /a{1, 3}/ - matches 'a' or 'aa' or 'aaa’

Special symbols:

- * previous symbol repeated from 0 to infinite times. Equivalent {0,}
- ? previous symbol repeated 0 or 1 times. Equivalent {0,1}
- + previous symbol repeated 1 to infinite times. Equivalent {1,}

RegExp Modifiers

g - global search, match all occurrences.
i - ignore case, match letters ignoring case.
m - multiline, match all lines, not line by line.

RegExp Symbol Classes

\d - digits
\D - non-digits
\w - word symbols [a-zA-Z0-9_]
\W - non-word symbols
\s - white-space symbols
\S - non white-space symbols

If you need to use some of the reserved symbol as match, you need to escape them with "\"
Reserved symbols: (,), \, /, *, +, ?, ., [,]

Further reading

RegExp:

Learning Regex The Hard Way

The Bastards Book Of Regular Expressions

RegExp Playground

Q&A

UA Frontend Online LAB