Regex

Regular Expressions

SAXION
UNIVERSITY OF
APPLIED SCIENCES

A **regular expression** (or **regex**)
is a special **sequence of characters**
used to **search for, match, or manipulate text**.

It acts like a powerful pattern-matching language.

History

- Began in the 1950's by Stephen Cole Kleene who studied algebraic notations to describe regular languages. This was purely theoretical

- In the 1960's and 1970's the theory was implemented by Ken Thompson in the ed tekst editor and other tools like grep

- 1980's - 2000, computing matured and regular expressions were integrated into many languages

- 2000's – now, Regex is used everywhere. They are used in web development, data validation, log analysis and much more

Why use Regular Expressions

- To find specific patterns in text (e.g., phone numbers, zip codes, email addresses)
- Validate input (e.g., check if a value is a valid email)
- Replace or remove certain patterns in text
- Extract data from “messy” strings

Formal definition of regular expressions

If A is an alphabet then the following are regular expressions:

- The empty string (ε)
- Any element of A
- If S and T are regular expressions

  then so are
  - ST (S followed by T)
  - S|T (S or T)
  - (S) (S between brackets to denote precedence)
  - S^* (0 or more occurrences of S)
  - S^+ (1 or more occurrences of S, superfluous)

In Practical Terms (Programmer's Definition)

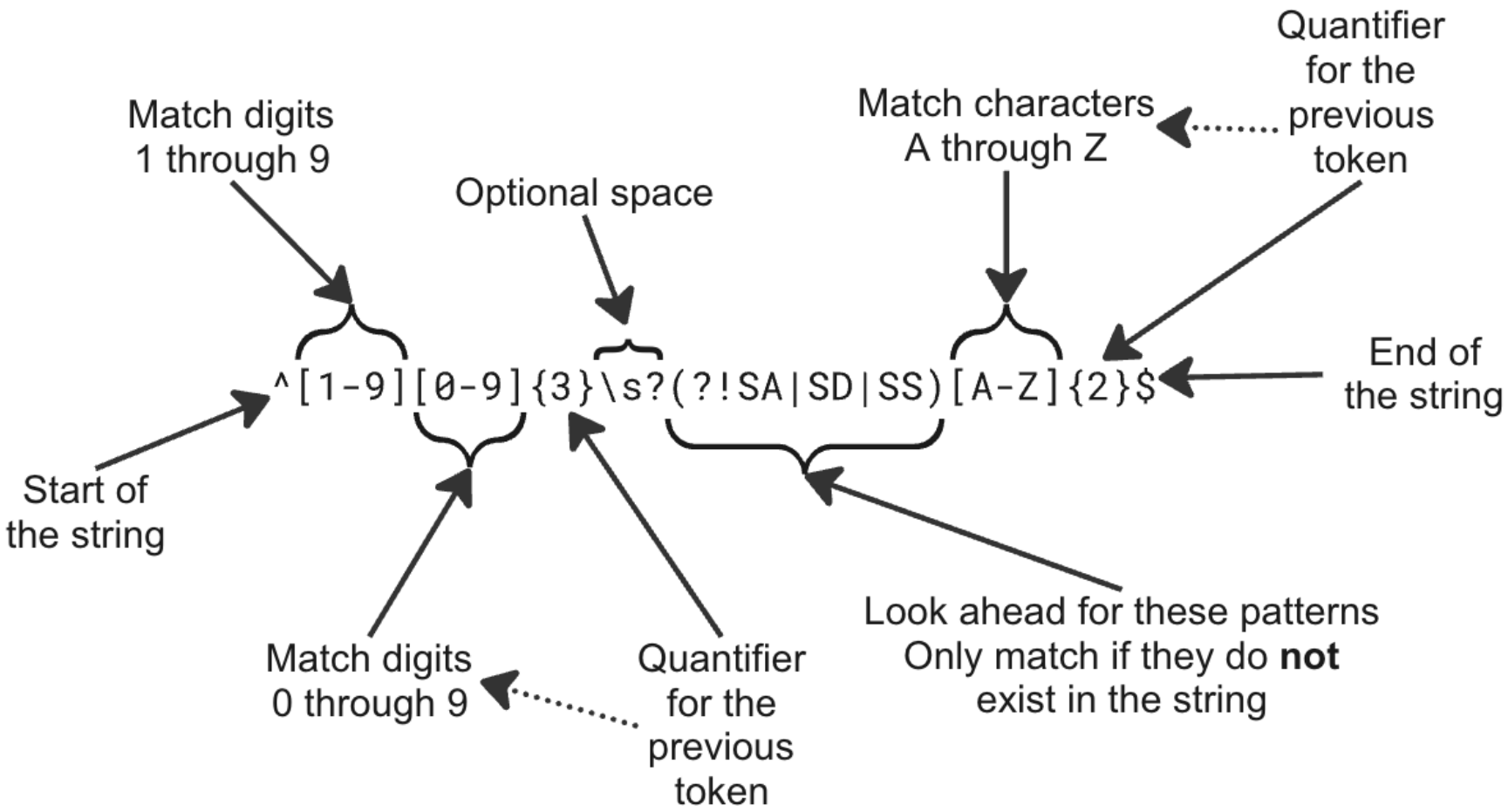
A **regular expression** is a pattern you write using special syntax to **match strings of text** based on certain rules (e.g., sequences, repetition, alternatives, etc.).

For example: A dutch zip code

- **4 digits** (cannot start with a 0)
- **Optional space** (officially there should be a space, but it's often omitted)
- **2 uppercase letters** (excluding the combinations **SA**, **SD**, and **SS**, which are not allowed)

```
^[1-9][0-9]{3}\s?(?!SA|SD|SS)[A-Z]{2}$
```

In more detail



Examples of regular expressions

Alphabet A = {a,b}	
a^*b^*	all strings with 0 or more a's, followed by 0 or more b's
a^*b^+	all strings with 0 or more a's, followed by 1 or more b's
$a^*\|b^*$	all strings with 0 or more a's, or 0 or more b's
(a|b)*	all strings with 0 or more a's or b's
a|b*	{ a, ε, b, bb, bbb, bbbb, … }
((aab)|(ab))*	{ ε, aab, aabaab, …, ab, abab, …, aabab, …, aabaababababaab, … }

ε == empty string

Regular expressions (1):

Pattern	Match	Example	Matches
*expr**	expr >=0 times	A*	ε, A, AA, AAA, …
expr+	expr >=1 times	A+	A, AA, AAA, …
expr?	expr 0 or 1 time	A?	ε, A
expr{*n*}	expr exactly n times	A{3}	AAA
expr{*n,m*}	expr >= n and <= m times	A{2,4}	AA, AAA, AAAA
expr{n,}	expr >= n times	A{2,}	AA, AAA, AAAA, AAAAA, …
exp1 | *exp2*	exp1 or exp2	A|B	A, B
char1 - *char2*	1 from range char1..char2	[a-z]	a, b, c, …, z
[*chars*]	1 from string “chars”	[a-z|A-Z|_][a-z|A-Z|0-9|_]*	Javascript identifiers
[^*chars*]	1 NOT from string “chars”	[^0-9]	A, a, #, %, ...

https://www.w3schools.com/java/java_regex.asp

Regular expressions (2):

Pattern	Match
.	any character
\s	whitespace (‘ ‘,’\t’,’\n’, …)
\S	non-whitespace
\d	digit; equals [0-9]
\D	non-digit; equals [^0-9]
\w	word character; equals [a-zA-Z_0-9]
\W	non-word character; equals [^a-zA-Z_0-9]
^expr	starting with expr
expr$	ending with expr

https://www.w3schools.com/java/java_regex.asp

Regular expressions

JAVASCRIPT

Creating regular expressions in Javascript

Regexes can be created in two ways

```
const regex = /abc/;
```

Or

```
const regex = new RegExp("abc");
```

Using regular expressions in Javascript (1)

Test is a string matches a regular expression

```
const regex = /^\d{4}\s?(?!SA|SD|SS)[A-Z]{2}$/;
regex.test("1234 AB"); // true
regex.test("1234 SS"); // false
```

Return the actual matches

```
const text = "My ZIP is 1234 AB.";
const match = text.match(/\d{4}\s?[A-Z]{2}/);
console.log(match[0]); // "1234 AB"
```

Using regular expressions in Javascript (2)

Replacing text

```
const dirty = "The code is 1234 SA.";
const clean = dirty.replace(/\d{4}\s?(SA|SD|SS)/, "XXXX XX");
console.log(clean); // "The code is XXXX XX."
```

Search for matches

```
const text = "Check 5678 ZZ.";
const index = text.search(/\d{4}\s?[A-Z]{2}/);
console.log(index); // 6
```

Exercises regular expressions

- Open https://regexr.com/ is a browser

- The site offers a lot of information about regular expressions

- More important though you can try and test regular expressions

- Create regular expressions for the excersies on the next slide

Time for exercises!

Exercises regular expressions

a) a Java (convention) class name
b) an integer number where preceding 0's are not allowed
c) a Dutch IBAN number which
   - starts with NL,
   - followed by 2 digits,
   - then a 4 upper case letter bank identification,
   - and then the internal account number of 10 digits.
d) an international IBAN, which is like a Dutch IBAN, but
   - starts with 2 uppercase letters,
   - has a single space before and after the bank identifier,
   - and the internal account number consists of 3 or more groups of 4 digits, each group separated by a space and the last group may contain less than 4 digits (but of course contains at least 1 digit).

Time for exercises!