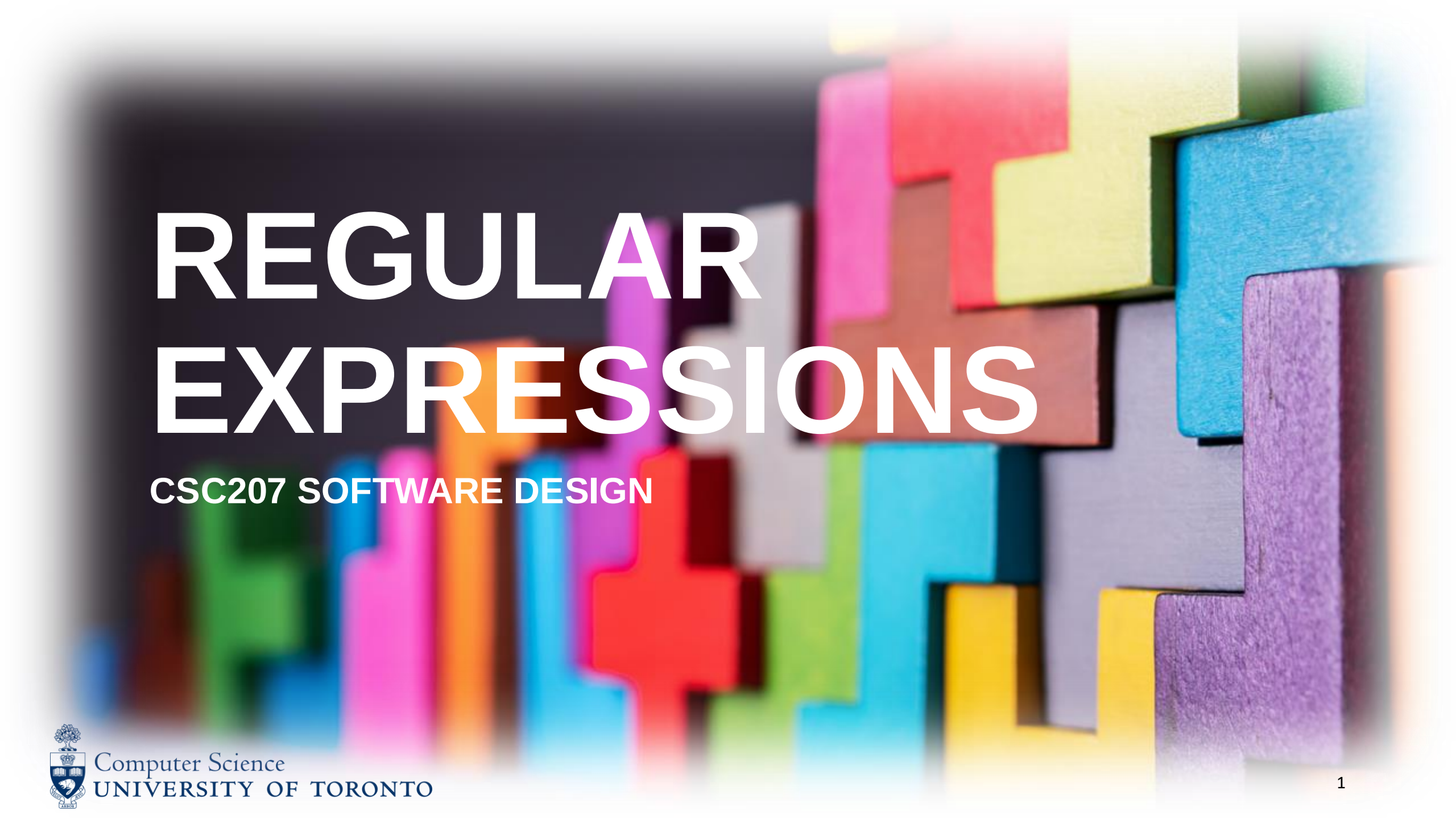

REGULAR EXPRESSIONS

CSC207 SOFTWARE DESIGN

Computer Science
UNIVERSITY OF TORONTO

LEARNING OUTCOMES

- Understand the basics of regular expressions (regex)
- Be able to develop regular expressions in Java.
- Be able to explain what expressions like those below mean:

```
^[a-z][a-zA-Z0-9]*$
[abc]C[a-e][24680]*[A-Z]
\(\d{3}\) \d{3}-\d{4}
^(([de])f)\2\1$
[a-t&&[r-z]]
```

WHAT IS A REGULAR EXPRESSION

- "a sequence of characters that specifies a match pattern in text"
- A given regex will match a set of strings.
- More on this in a bit, but first, what can we *do* with regular expressions?

More about the history + an overview of regex if you are interested:
https://en.wikipedia.org/wiki/Regular_expression

REGEX QUESTIONS

- Regular expressions can be used to find data in documents:
  - Find phone numbers (in a file, on a webpage)
  - Find email addresses
  - Find CSC course codes
  - Find (and replace) in IntelliJ!
- They can also be used to check for validity:
  - Is a password strong enough with the right set of characters?
  - Does a variable name conform to the Java style guidelines?

Extracting **substrings** which match a pattern

Does a given string match a pattern?

DESCRIBING A SET OF STRINGS

- IDEs like IntelliJ might describe the Java naming conventions for variables this way:

```
^[a-z][a-zA-Z0-9]*$
```

- This is a *regular expression* (or *regex*)
- This describes a pattern that appears in a set of strings. We say that any such string *matches* or *satisfies* the regular expression.
- “A string which matches this regex is consistent with the Java naming conventions for variables”
- Aside: you can see lots of similar examples in the mystyle.xml configuration file which we have used for Checkstyle this term!

^[a-z][a-zA-Z0-9]*$

- The `^` character means that the pattern must start at the beginning of the string. This is called an *anchor*.
- Square brackets `[]` tell you to choose one of the characters listed inside.
  - In the leftmost set of brackets, we are given all lowercase English letters to choose from.
  - The second character will come from the second set of square brackets. It can be any lowercase letter, uppercase letter, or digit.
- The `*` means zero or more of whatever immediately precedes it. This is an example of a *quantifier*.
- The `$` signifies the end of the string. This is another anchor.

^[a-z][a-zA-Z0-9]*$ continued…

- So, our entire string must consist of letters and numbers, with the first character being a lower-case letter.
- Here are some examples:
  - `x, numStudents, obj1`
- These do not satisfy the regular expression. (Why not?)
  - `Alphabet, 2ab, next_value`
- Do any of these strings match?
  - `z3333, aBcB041, 78a`

WHAT IF THERE ARE NO ANCHORS?

- Here is another regular expression:
  - `[abc]C[a-e][24680]*[A-Z]`

- When this regex is applied to a string, it will find the substrings that match.
  - `cCaA` matches the entire expression
  - `ABCcCcCa1A23` contains substrings that match. For example, `cCcC` matches but not `cCa1A`.

SPECIAL SYMBOLS

- A period `.` matches any character.
- Whitespace characters (the backslash is the escape character)
  - `\s` is any whitespace character
  - `\t` is a tab character
  - `\n` is a new line character
- Just inside a square bracket, `^` has another meaning: it matches any character *except* the contents of the square brackets.
  - For example, `[^aeiouAEIOU]` matches anything that isn't a vowel.

CHARACTER CLASSES (MORE ESCAPES)

- You can make your own character classes by using square brackets like `[q-z]`, `[AEIOU]`, and `[^1-3a-c]`, or you can use a predefined class.

Construct	Description
.	any character
\d	a digit [0-9]
\D	a non-digit [^0-9]
\s	a whitespace char [\t\n\x0B\f\r]
\S	a non-whitespace char [^\s]
\w	a word char [a-zA-Z_0-9]
\W	a non-word char [^\w]

QUANTIFIERS

- `*` means zero or more, `+` means one or more, and `?` means zero or one.

- We append `{2}` to a pattern for exactly two copies of the same pattern, `{2,}` for two or more copies of the same pattern, and `{2,4}` for two, three, or four copies of the same pattern.

Pattern	Matches	Explanation
a*	‘’ ‘a’ ‘aa’	zero or more
b+	‘b’ ‘bb’	one or more
ab?c	‘ac’ ‘abc’	zero or one
[abc]	‘a’ ‘b’ ‘c’	one from a set
[a-c]	‘a’ ‘b’ ‘c’	one from a range
[abc]*	‘’ ‘acbccb’	combination

ESCAPING A SYMBOL

- Sometimes we want symbols to show up in the string that otherwise have meanings in regular expressions. To “escape” the meaning of the symbol, we write a backslash `\` in front of it.
- A period `.` means any character. To have a period show up in the string, we write `\.`
- Ex 1: `abc123` matches the regex `[a-e][a-e].+`
- Ex 2: `1.4` matches the regex `[0-9]\.[0-9]`

REPETITION OF A PATTERN VS. A SPECIFIC CHOICE OF CHARACTER

- Here is a pattern that describes all phone numbers on the same continent:
- `\(\d\d\d\) \d\d\d-\d\d\d\d`
- `\((\d\d\d)\) \d\d\d-\d\d\d\d`
- We could also write this as
- `\(\d{3}\) \d{3}-\d{4}`
- Example: `(123) 456-7890` matches the pattern.
- The above `\d{3}` repeated the `\d` pattern three times, but what if we wanted to repeat the *exact* same digit three times instead?
  - We can do this too — with some additional syntax!

REPETITION OF EXACT CHARACTERS

- To repeat the same character twice, we use **groups** which are denoted by round brackets. Then we escape the number of the group we want to repeat:
- The string `124124a124` matches the regular expression:
- `(\d\d\d)\1a\1`
- Groups are assigned the number of open brackets that precede them.
- For example, `^(([de])f)\2\1$` will repeat both groups. The strings that match are:
  - `dfddf` and `efeef`
- Group 1 corresponds to `[de]f` and Group 2 corresponds to `[de]`
  - `df d df` `ef e ef`

LOGICAL OPERATORS

- `|` means “or”
- `&&` means the intersection of the range before the ampersands and the range that appears after. For example `[a-t&&[r-z]]` would only include the letters `r`, `s`, and `t`.
- Note: different implementations of regex may support slightly different operators.

Construct	Description
[abc]	a, b, or c (simple class)
[^abc]	any char except a, b, or c (negation)
[a-zA-Z]	a through z or A through Z inclusive (range)
[a-d[m-p]]	a through d or m through p (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z except for b and c (subtraction)
[a-z&&[^m-p]]	a through z and not m through p (subtraction)

ANCHORS EXAMPLE

Pattern	Text	Result
b+	abbc	Matches
^b+	abbc	Fails (no b at start)
^a*$	aabaa	Fails (not all a's)

REGEX IN JAVA

- The `String` class
  - `split`, `matches`, `replaceAll`, and `replaceFirst`
- The `Pattern` class
  - https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html
- The `Matcher` class
  - https://docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html
- See RegexMatcher.java and RegexChecker.java on Quercus for small demos using these classes and some of their methods.

OTHER RESOURCES

- Quick reference
  - http://www.rexegg.com/regex-quickstart.html
- Tutorial specific to Java
  - http://tutorials.jenkov.com/java-regex/index.html
- Regex crossword
  - https://regexcrossword.com
- One of many online interfaces to experiment with regex
  - https://regex101.com