

# Regular Expressions

- A *Regular Expression (regex)* is a pattern that can be matched against a string



"999423210"



Regular  
Expression

"718-997-5000"



Regular  
Expression

"k1ord@qc.cuny.edu"



Regular  
Expression

```
import java.util.regex.*;

public static boolean isValidSSN(String ssn)
{
    Pattern p;
    Matcher m;
    String SSN_PATTERN = Regular Expression;
    p = Pattern.compile(SSN_PATTERN);
    m = p.matcher(ssn);
    return matcher.matches();
}
```

# Constants

## Match exactly the string inside the regex

```
public static boolean isValidSSN(String ssn) {  
    Pattern p;  
    Matcher m;  
    String SSN_PATTERN = "9999999999";  
    p = Pattern.compile(SSN_PATTERN);  
    m = p.matcher(ssn);  
    return matcher.matches();  
}
```

`isValid("9999999999")`      returns true 😊

`isValid("9999999998")`      returns false ☹

# Character classes

match any character inside [ ]

[abc]	a, b, or c (simple class)
[^abc]	Any character 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: [a-dm-p] (union)
[a-z&&[def]]	d, e, or f (intersection)
[a-z&&[^bc]]	a through z, except for b and c: [ad-z]
[a-z&&[^m-p]]	a through z, and not m through p: [a-lq-z]

# Character Range

```
public static boolean isValidSSN(String ssn) {  
    Pattern p;  
    Matcher m;  
    String SSN_PATTERN = "[0-9]";  
    p = Pattern.compile(SSN_PATTERN);  
    m = p.matcher(ssn);  
    return matcher.matches();  
}
```

<code>isValid("999999999")</code>	returns true	😊
<code>isValid("999999998")</code>	returns true	😊
<code>isValid("99a")</code>	returns true	😞

# Predefined Character Classes

- . Any character (may or may not match line end)
- \d A digit: [0-9]
- \D A non-digit: [^0-9]
- \s A whitespace character: [ \t\n\x0B\f\r]
- \S A non-whitespace character: [^\s]
- \w A word character: [a-zA-Z\_0-9]
- \W A non-word character: [^\w]

# [0-9] is the same as \d

```
public static boolean isValidSSN(String ssn) {  
    Pattern p;  
    Matcher m;  
    String SSN_PATTERN = "\\d";  
    p = Pattern.compile(SSN_PATTERN);  
    m = p.matcher(ssn);  
    return matcher.matches();  
}
```

<code>isValid("999999999")</code>	returns true	😊
<code>isValid("999999998")</code>	returns true	😊
<code>isValid("99a")</code>	returns true	😞



# Quantifiers

$X?$

$X$ , once or not at all

$X^*$

$X$ , zero or more times

$X^+$

$X$ , one or more times

$X\{n\}$

$X$ , exactly  $n$  times

$X\{n,\}$

$X$ , at least  $n$  times

$X\{n,m\}$

$X$ , at least  $n$  but not  
more than  $m$  times

# `\d{9}` matches 9 digits

```
public static boolean isValidSSN(String ssn) {  
    Pattern p;  
    Matcher m;  
    String SSN_PATTERN = "\\d{9}";  
    p = Pattern.compile(SSN_PATTERN);  
    m = p.matcher(ssn);  
    return matcher.matches();  
}
```

<code>isValid("999999999")</code>	returns true 😊
<code>isValid("999999998")</code>	returns true 😊
<code>isValid("99a")</code>	returns false 😊
<code>isValid("SSN is 999999999")</code>	returns true 😞

# $\wedge$ = beginning of regex, $\$$ = end

```
public static boolean isValidSSN(String ssn) {  
    Pattern p;  
    Matcher m;  
    String SSN_PATTERN = " $\wedge$ \\d{9}$";  
    p = Pattern.compile(SSN_PATTERN);  
    m = p.matcher(ssn);  
    return matcher.matches();  
}
```

`isValid("999999999")` returns true 😊

`isValid("999999998")` returns true 😊

`isValid("99a")` returns false 😊

`isValid("SSN is 999999999")` returns false 😊

`isValid("999-99-9999")` returns false ☹

# Reasonable regex for SSN

```
public static isValidSSN(String ssn) {  
    Pattern p;  
    Matcher m;  
    String SSN_PATTERN = "^\\d{3}-?\\d{2}-?\\d{4}$"  
    p = Pattern.compile(SSN_PATTERN);  
    m = p.matcher(ssn);  
    return matcher.matches();  
}
```

```
/* Uses split to break up a string of input separated by
 * commas and/or whitespace.
 */
import java.util.regex.*;
public class Splitter {
    public static void main(String[] args) {
        String myString = "one,two, three    four ,   five";

        Pattern p = Pattern.compile("[,\\s]+");

        String[] result = p.split(myString);

        for (int i=0; i<result.length; i++)
            System.out.println(result[i]);
    }
}
```

# Find numbers

```
import java.util.regex.*;
public class FindNumbers {
    public static void main(String[] args) {
        String myString =
            "hello, this 123 is 5643 testz w123ith 5 words";
        Pattern p = Pattern.compile("[0-9]+");
        Matcher m = p.matcher(myString);
        while (m.find()) {
            System.out.println(m.group());
        }
    }
}
```

Output:

123

5643

123

5

# Capturing what is matched

```
import java.util.regex.*;
public class TelephoneValidation {
    public static void main (String[] args) {
        Pattern p;
        Matcher m;
        String Tel_Pattern = "^((\\d{3})-?\\d{3}-?\\d{4})$";
        p = Pattern.compile(Tel_Pattern);
        m = p.matcher("718-997-5000");
        if(m.find()) {
            System.out.println(m.group(0));
            System.out.println(m.group(1));
        }
    }
}
```

Output:

718-997-5000


718

```

public static void main (String[] args) {
    String[] phoneNums = {"718-997-5000", "7189975000",
                          "718.997.5000", "718-997.5000"};

    Pattern p;
    Matcher m;
    String Telephone_Pattern =
        " (\\d{3}) ([\\.-]?) \\d{3} \\2 \\d{4}";
    p = Pattern.compile(Telephone_Pattern);
    for (int i=0; i<phoneNums.length; i++) {
        m = p.matcher(phoneNums[i]);
        if (m.matches())
            System.out.println(phoneNums[i]+" Matches.");
        else
            System.out.println(phoneNums[i]+
                               " does not match.");
    }
}

```



718-997-5000 Matches.  
 7189975000 Matches.  
 718.997.5000 Matches.  
 718-997.5000 does not match.



# Password Validation

```
((?=.*\d) (?=.*[a-z]) (?=.*[A-Z]) (?=.*[@#$%]) .{6,20})
```

```
(                # Start of group
(?=.*\d)         # must contain one digit from 0-9
(?=.*[a-z])      # must contain one lowercase character
(?=.*[A-Z])      # must contain one uppercase character
(?=.*[@#$%])     # must contain one special symbol "@#$%"
.                # match anything else
{6,20}           # length at least 6 and maximum 20
)                # End of group
```

# Validate Email Address

```
^[_A-Za-z0-9-]+(\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+(\.[A-Za-z0-9]+)*(\.([A-Za-z]{2,}))$
```