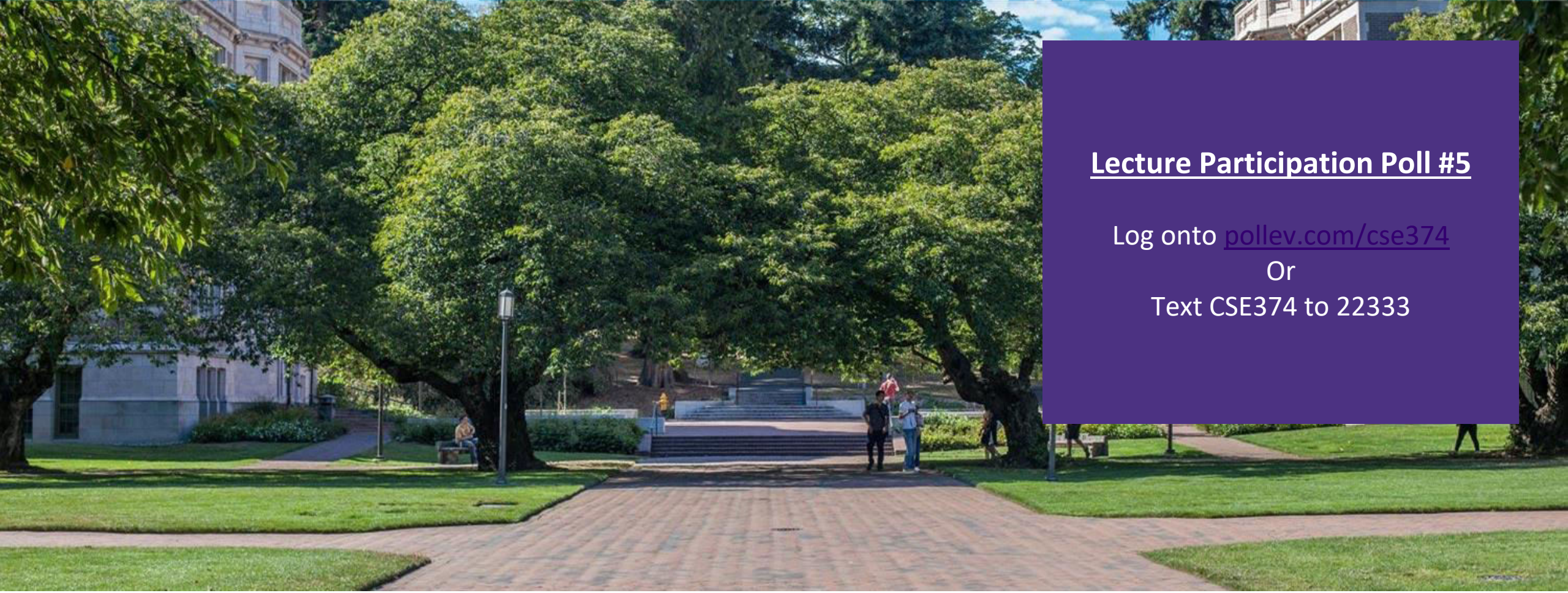

Lecture Participation Poll #5

Log onto pollev.com/cse374
Or
Text CSE374 to 22333

Lecture 6: Regex

CSE 374: Intermediate Programming Concepts and Tools

Administrivia

Sorry the poll everywhere closed over the weekend

Try changing your password by just typing passwd (no username)

Gradescope Add Code: ERPPB2

Office Hours posted!
https://courses.cs.washington.edu/courses/cse374/21au/oh/
(click "week" for easier view)

Regular Expressions!

```
/^[a-zA-Z_\-]+@(([a-zA-Z_\-])+\.)+[a-zA-Z]{2,4}$/
```

Regular expression ("regex"): a description of a pattern of text

- Can test whether a string matches the expression's pattern
- Can use a regex to search/replace characters in a string

Regular expressions are extremely power but tough to read (the above regular expression matches email addresses)

Regular expressions occur in many places:

- Java: Scanner, String's `split` method (CSE 143 random grammar generator)
- Supported by HTML5, JS, Java, Python, PHP, and other languages
- Many text editors (TextPad, Sublime, Vim, etc.) allow regexes in search/replace
- The site Rubular is useful for testing a regex

https://regex101.com/ https://regexcrossword.com/ https://regexone.com/

Glob patterns

- Syntax to replace a pattern with a list of file names that all match that pattern
  - Enables you to pass multiple file names as arguments without typing them out individually
  - Pattern matches are based on location within file directory
- Wildcard – * – anything goes here
  - EX: echo src/*
  - Src/file1.txt src/file2.txt src/file3.txt
  - Example uses
    - echo * – prints every file/folder in current directory
    - echo *.txt – finds all files with that extension within directory
    - echo /bin/python* – finds all files within that path because they start with that string
    - cp src/* dest/ – copies all files from one directory to another
    - find –name '*.txt' recursively finds files ending in .txt

Basic Regular Expression

```
/abc/
```

The simplest regexes simply match a particular substring
The above regular expression matches any string containing "abc"

- Match: "abc", "abcdef", "defabc", ".=.abc.=.", ...
- Don't Match: "fedcba", "ab c", "PHP", …

Wildcards, Case sensitivity

A `.` matches any character except a `\n` line break

- `/.ax../` matches "Faxes", "Jaxes", "Taxes", "maxie", etc.

A trailing i at the end of a regex (after the closing /) signifies a case-insensitive match

- `/cal/i` matches "Pascal", "California", "GCal", etc.

Quantifiers: *, +, ?

* means 0 or more occurrences
  - `/abc*/` matches "ab", "abc", "abcc", "abccc", ...
  - `/a(bc)*/` matches "a", "abc", "abcbc", "abcbcbc", ...
  - `/a.*a/` matches "aa", "aba", "a8qa", "a!?xyz__9a", ...

+ means 1 or more occurrences
  - `/Hi!+ there/` matches "Hi! there", "Hi!!! there!", ...
  - `/a(bc)+/` matches "abc", "abcbc", "abcbcbc", ...

? means 0 or 1 occurrences
  - `/a(bc)?/` matches only "a" or "abc"

Regex special characters

`\` - escape following character

`.` – matches any single character at least once

- `c.t` matches `{cat, cut, cota}`

`|` - or, enables multiple patterns to match against

- `a|b` matches `{a}` or `{b}`

`*` - matches 0 or more of the previous pattern (greedy match)

- `a*` matches `{, a, aa, aaa, …}`

`?` – matches 0 or 1 of the previous pattern

- `a?` matches `{, a}`

`+` - matches one or more of previous pattern

- `a+` matches `{a, aa, aaa, …}`

`{n}` – matches exactly n repetitions of the preceding

- `a{3}` matches `{aaa}`

`()` – groups patterns for order of operations

`[]` – contains literals to be matched, single or range

- `[a-b]` matches all lowercase letters

`^` - anchors to beginning of line

- `^//` matches lines that start with `//`

`$` - anchors to end of line

- `;$` matches lines that end with `;`

Character ranges: [start-end]

Inside a character set, specify a range of characters with -

- `/[a-z]/` matches any lowercase letter
- `/[a-zA-Z0-9]/` matches any lowercase or uppercase letter or digit

Inside a character set, - must be escaped to be matched

- `/[+\-]?[0-9]+/` matches an optional + or -, followed by at least one digit

Practice: Write a regex for Student ID numbers that are exactly 7 digits and start with a 1

```
-- Pass --
1234567

-- Fail --
7654321
123abcd
123
```

```
1[0-9]{6}
```

grep with options

```
grep [options] [pattern] [file]

grep -c "string" FILENAME # -c count number of matches

grep -i "string" FILENAME # -i case insensitive

grep -w "string" FILENAME # -w checks for words and not substrings

grep -A <N> "string" FILENAME # -A prints N lines after match

grep -B <N> "string" FILENAME # -B prints N lines before match

grep -r "string" * # -r recursive search from current directory
```

grep with regex

- Useful Regex Patterns
  - [a-zA-Z] - matches all English letters
  - [0-9]* - matches list of numbers
  - (abc)* - match any number of “abc”s
  - (foo | bar) – matches either “foo” or “bar”

```
-grep “^hello” file1 #Match all lines that start with ‘hello’
-grep “done$” file1 #Match all lines that end with ‘done’
-grep “[a-e]” file1 #Match all lines that contain any of the letters a-e
-grep “ *[0-9]” file1 #Match all lines that start with a digit following
 zero or more spaces. E.g: “ 1.” or “2.”
```

Extended Regex

grep –E uses “extended” regex

- In basic regular expressions the meta-characters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions \?, \+, \{, \|, \(, and \).
- Traditional egrep did not support the { meta-character, and some egrep implementations support \{ instead, so portable scripts should avoid { in grep -E patterns and should use [{] to match a literal {.
- Also grep -e allows to use several strings for searching: 'grep -e 'abc' -e 'def' -e '123' will look for any of the three of these strings: abc as well as def and 123.

```
grep -E '^[A-Z].*[.,]$' file.txt
```

- match all lines that start with a capital letter and end with either period or comma
- .* matches any number of any character

https://stackoverflow.com/questions/17130299/whats-the-difference-between-grep-e-and-grep-e

Grep regex demo