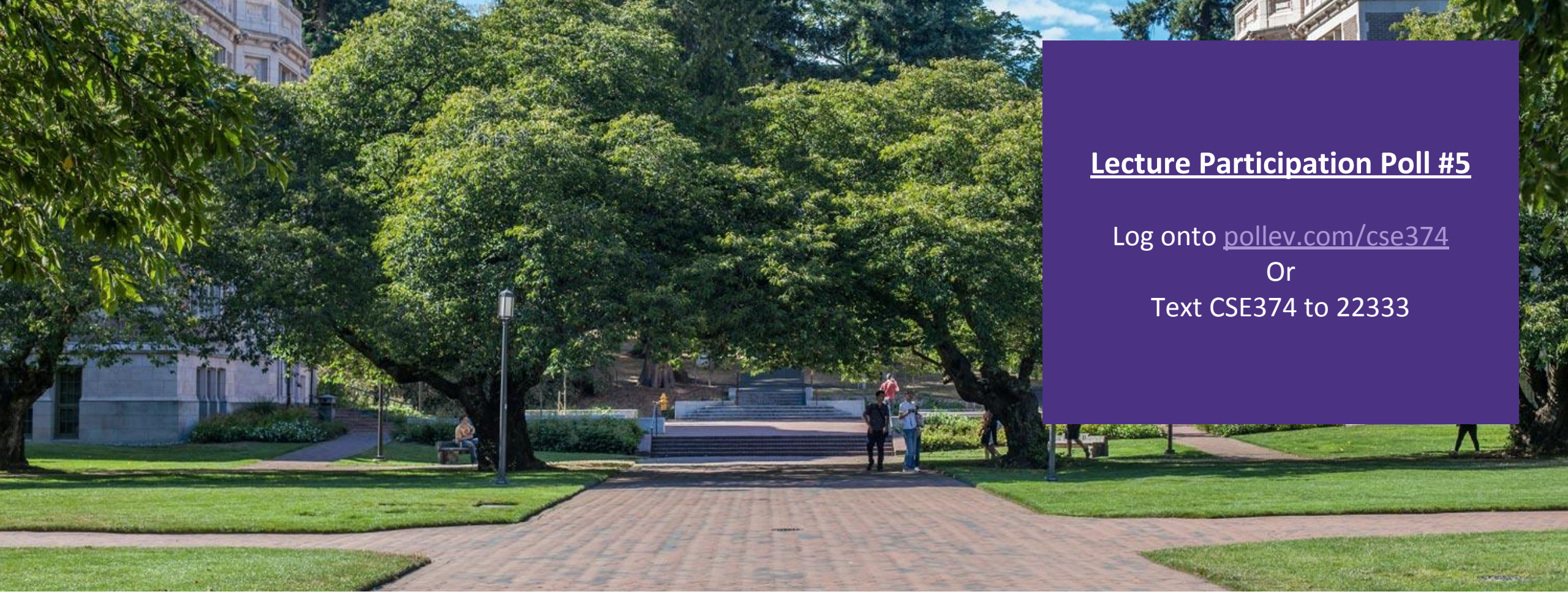

Lecture Participation Poll #5

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

Lecture 6: Regex

CSE 374: Intermediate Programming Concepts and Tools

Administrivia

Bash Script Variables

- When writing scripts you can use the following default variables

$# - stores number of parameters entered

Ex: `if [$# -lt 1]` tests if script was passed less than 1 argument

$N - returns Nth argument passed to script

Ex: `sort $1` passes first string passed into script into sort command

$0 – command name

Ex: `echo "$0 needs 1 argument"` prints "<name of script> needs 1 argument"

$* returns all arguments

$@ returns a space separated string containing all arguments

"$@" prevents args originally quoted from being read as multiple args

Control Flow in bash

- Bash has loops and conditionals like most languages
- If Statements

```
if <test> then
  <commands>
fi
```

Ex:

```
if ./myprogram args; then
  echo "it works!"
else
  echo "it didn't work"
fi
```

Executes body if ./myprogram succeeds (returns exit code 0)

- For loop

```
for <variable> in <list>
do
  <commands>
done
```

Ex:

```
for word in "list of words"
fo
  echo $word
done
```

"lists" in bash are just strings with white space separators

- while loop

```
while [test] do
  <commands>
done
```

Conditionals

- Test evaluates Boolean comparison of two arguments

```
test "$str1" == "$str2" #tests string equality
test -f result.txt #checks if file exists with -f option
test $num -eq 0 #checks integer equality with -eq option
test $# -ne 2 #checks if ints are not equal with -ne option
```

  - Other useful options: -lt -le -gt -ge

- Combine test with if by replacing "test" with []

```
if [ -f result.txt ]; then
```

  - Spaces around the brackets and semicolon are required

- Bash understands Boolean logic syntax
  - && and
  - || or
  - ! not

Common If Use Cases

- If file contains

```
if grep –q –E ‘myregex’ file.txt; then
  echo “found it!”
fi
```

-q option “quiet” suppresses the output from the loop

If is gated on successful command execution (returns 0)

- If incorrect number of arguments passed

```
if [ $# –ne 2 ]; then
  echo “$0 requires 2 arguments” >&2
  exit 1
fi
```

Checks if number of arguments is not equal to 2, if so prints an error message to stderr and exits with error code

Common loop use cases

- Iterate over files

```
for file in $(ls)
do
  if [-f $file ]; then
     echo "$file"
  fi
done
```

<- All files + directories

- Iterate over arguments to script

```
while [ $# -gt 0 ]
do
  echo $*
  shift
done
```

Shift command moves through list of arguments

Similar to .next in Java Scanner

Exit Command

- Ends a script's execution immediately
  - Like "return"
- End scripts with a code to tell the computer whether the script was successful or had an error
- 0 = successful
  - exit without a number defaults to 0

```
exit
exit 0
```

- Non 0 = error

```
exit 1
```

Scripting demo: combine

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

Regex

- Regular expressions (regex) are a set of rules for matching patterns in text
  - Used across programming languages and math
  - Different applications might have slightly different rules (yeah, it's frustrating…)
- Regex patterns can include characters, anchors and modifiers
  - Characters = the literal characters you are trying to match
  - Anchors – set the position in the line where a pattern may be found
    - ^ anchor to front
    - $ anchor to end
  - Modifiers – modify the range of text pattern can match
    - * matches any number of characters
    - [set of chars]
- Regex basics, let P be our pattern and S be a string to match
  - P can be a single character (ex: a) to match S of the same single character
  - P_1P_2 matches S if $S=S_1S_2$ where $P_1 = S_1$ and $P_2 = S_2$
  - $P_1|P_2$ matches S if P1 or P2 matches S
- grep –e finds using regex
  - By default grep matches against .*p.*

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

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 `;`

Useful 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 and regex

- grep –e uses “extended” regex

Grep regex demo