Regular Expressions

Backreferences, Accessing Matches, and Substitution

Warm-Up

In [ ]:

```
# Write a regular expression that finds all songs
# in the top 100 by only one person
# Finesse by Bruno Mars & Cardi B SHOULD NOT MATCH
# Too Good At Goodbyes by Sam Smith SHOULD MATCH
# Young Dumb & Broke by Khalid SHOULD MATCH
#


foreach my $s (@songs) {
  say $s if $s =~ /REGEX_HERE/;
}
```

Today's Data

- Today we will be working CSV data of international airports and their cities, countries, and continent
- Each line has the format of

```csv
City, Airport Name, Airport Code, Country, Continent
```

- The data was scraped from https://en.wikipedia.org/wiki/List_of_international_airports_by_country

Backreferences

- Last lecture we primarily used grouping to make our regex's neater.
- One of the most powerful uses of grouping is to specify seeing the same match later in the expression
- Each group is assigned a number by the regular expression engine
  - To refer back to that group, use backslash followed by the number, e.g. \1

In [ ]:

```
open(my $fh,  'data/airports.tsv');

while (my $row = <$fh>) {
   chomp $row; #Remove trailing \n
   say $row if $row =~ /^.*\t(\b\w+\b).*\W.*\1.*\t.*\t.*\t.*$/;
}
close($fh);
```

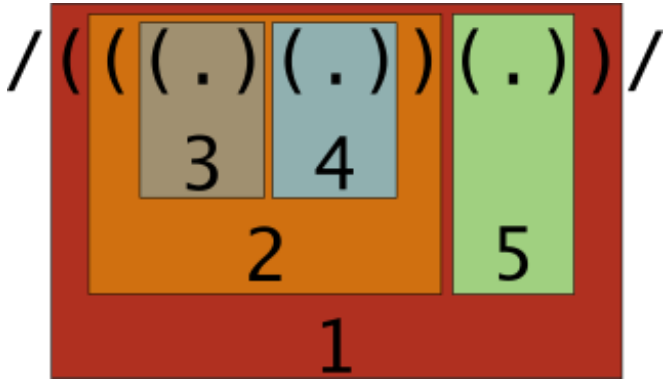
In [ ]:

```
open(my $fh,  'data/airports.tsv');

while (my $row = <$fh>) {
   chomp $row; #Remove trailing \n
   say $row if $row =~
       /\t.*(\w).*(\w).*(\w).*\t\1\2\3\t.*\t.*$/;
}
close($fh);
```

Backreference Ordering

- If there are multiple groups in a regex, they are numbered by their left parentheses
- This can get confusing, here is a helpful chart presented by Dan Hood



Backrefercing Practice

In [ ]:
```
# Write a regular epxression that finds airports with at
# least part of their country name in the airport name.
# Alternatively, find a country with part  of the airport
# name in it
open(my $fh,  'data/airports.tsv');

while (my $row = <$fh>) {
  chomp $row; #Remove trailing \n
  say $row if $row =~ /REGEX_HERE/;
}
close($fh);
```

Accessing Matches

- Often we want to retrieve a specific part of the match
- We can do this by using groups, and then referring back to the group number later in the code
- Each language has a slightly different way of doing this
  - In Perl this uses the same numbering scheme as back-references, but the matches are stored in Perl variables
  - If it is the first match, use `$1` rather than `\1`

In [ ]:

```
open(my $fh,  'data/airports.tsv');

while (my $row = <$fh>) {
  chomp $row; #Remove trailing \n
  if ($row =~ /\t(.*)\t.*\tEngland/){
      say $1;
  }
}
close($fh);
```

In [ ]:

```
open(my $fh,  'data/airports.tsv');

while (my $row = <$fh>) {
  chomp $row; #Remove trailing \n
  if ($row =~ /^(.*)\t\1(.+)\sInternational Airport\t/){
      say $2;
  }
}
close($fh);
```

Substitution Introduction

- Many times the reason we want to know if something is present is so we can replace it
- Regular expressions give us much more powerful, dynamic ways of replacing than just string literals offer
- The following aren't possible (or aren't simple) with string literals
  - 410-455-1000 → (410) 455-1000
  - `if x == 4 : print x , y ; x , y = y , x` → `if x == 4: print x, y; x, y = y, x`

Substitution Basics

- In Perl the syntax for a substitution regex is `s/regex/substitution/`
- The regex is the only part that can use metacharacters
  - The substition can consist of literal characters or special variables

In [ ]:
```
$ssn = "A social security number looks like 000-12-3456 or 000-98-7654";
$ssn =~ s/\d{3}-\d\d-\d{4}/****/;
say $ssn;
```

Simple Substitution using the g Modifier

- In most cases, we want to use substitution to substitute all matches, so we should use the g modifier

In [ ]:
```
$ssn = "A social security number looks like 000-12-3456 or 000-98-7654";
$ssn =~ s/\d{3}-\d\d-\d{4}/****/g;
say $ssn;
```

Simple Substitution with Literals

- The pattern portion can consist only of literals
  - Many languages now have a specific replace method or function to operate on strings
  - Still very useful to use fast simple tools like `sed`

In [ ]:
```
$umbc = "UMBC is located in MD";
$umbc =~ s/UMBC/The University of Maryland, Baltimore County/g;
say $umbc;
$umbc =~ s/MD/Maryland/g;
say $umbc;
```

Basic Substitution Practice

In [ ]:

```
## Write a substitution pattern to replace any non-legal UNIX filename
# characters with an underscore. Multipe non-legal characters in
# a row should be replaced with a single underscore
# Legal Characters: A-Z, a-z, 0-9, . , - , _
$file_name = " My invalid / file[Name]";
$file_name =~ s/REGEX/REPLACE/g;
say $file_name;
```

Backreference Variables

- Many common tasks, like reformatting, involving saving part of the match
  - To refer to a group found in the pattern, use $x, where x is the group number

In [ ]:
```
$today = "Today's date is 2-5-18";
$today =~ s/(\d?\d)-(\d?\d)-(\d\d)/$1\/$2\/$3/g;
say $today;
```

In [ ]:
```
$today = "Today's date is 02-05-18";
$today =~ s/(\d?\d)-(\d?\d)-(\d\d)/$1\/$2\/$3/g;
say $today;
```

In [ ]:
```
$ssn = "A social security number looks like 000-12-3456 or 000-98-7654";
$ssn =~ s/\d\d\d-\d\d-(\d{4})/***-**-$1/g;
say $ssn;
```

Sidenote: Changing Delimiters

- When matching or substituting a string with the / character, it can be very annoying to escape all of them
- Almost any puncuation can be used as the delimiter
  - If it is a character that comes in pairs, you should use the left and right versions

In [ ]:
```
$today = "Today's date is 02-05-18";
$today =~ s[(\d?\d)-(\d?\d)-(\d\d)][$1/$2/$3]g;
say $today;
```

In [ ]:

```
$today = "Today's date is 09-07-17";
$today =~ s!(\d?\d)-(\d?\d)-(\d\d)!$1/$2/$3!g;
say $today;
```

Substitution Live Example

In [ ]:
```
# Given a string of non PEP compliant spacing in
# [] or {} or (), remove all extraneous spacing
# array[ 4 ] -> array[4]
$x = "spam( ham[ 1 ], { eggs: 2 } )";
$x =~ s/REGEX/SUBSTITUTE/g;
say $x;
```

Substitution Practice

In [ ]:
```
# Repalace all relative links (of the form href="index.html" etc,)
# with absolute links, (of the
#form href="https://cs.umbc.edu/coursese/undergraduate/433/index.html")
# assume absolute path is as above

$html = '<a href="index.html">A link</a><a href="img/my_img.png">An image</a>';
$html =~ s|REGEX|SUBSTITUTE|g;
say $html;
```

Lookahead and Lookbehind

- In some instances, we want to match, but not capture a piece of text
  - Are zero-width assertaions
  - After the look ahead is complete we return to the same place in the text

- A lookahead is written as:

  `(?=pattern)`

- A lookbehind is written as:

  `(?<=pattern)`

  - They usually cannot be variable length

Lookahead Example

In [ ]:

```
open(my $fh,  'data/airports.tsv');

while (my $row = <$fh>) {
  chomp $row; #Remove trailing \n
  if ($row =~ /\t.*\t(?=.*\tEngland)/){
      say $&;
  }
}
close($fh);
```

In [ ]:

```
open(my $fh, 'data/airports.tsv');

while (my $row = <$fh>) {
  chomp $row; #Remove trailing \n
  if ($row =~ s/\t(.*)\t(?=.*\tEngland)/\tBritian's $1\t/){
      say $row;
  }
}
close($fh);
```

Lookbehind Example

In [ ]:

```
open(my $fh, 'data/airports.tsv');

while (my $row = <$fh>) {
  chomp $row; #Remove trailing \n
  if ($row =~ /(?<=London\t).*/){
      say $&;
  }
}
close($fh);
```

Lookahead and Lookbehind Example

- Lets assume that in our text every 7 digit number is a phone number

In [ ]:
```
$bad_number = "1234567";
$bad_number =~ s/(?<=\d\d\d)(?=\d\d\d\d)/-/g;
say $bad_number;
```

Live Lookahead Example

- In the beginning of the lecture we looked at how regex's are helpful for code reformattting
- The specific python convention we looked at was no spaces immediately before a comma, semicolon, or colon

In [ ]:
```
$code = "if x == 4 : print x , y ; x , y = y , x"
$code =~ s/REGEX/REPLACEMENT/g;
say $code;
```

Lookahead Practice

In [ ]:

```
# Substitute all instances of James with President,
# when followed by  Monroe or Madison or Polk

$text = <<HERE;
James is a common name for presidents, there have been many
presidents named James, like James Madison, James Monroe,
and James Polk. Lebron James has not been US president.
HERE

$text =~ s/REXEG/SUBSTITUTION/g;
say $text;
```

Negative Lookahead and Behind

- A useful ability is to ensure the thing you are looking for is not followed or preceded by something
- This is a negative lookahead or lookbehind, and the syntax is almost identical, except the = is now a `!`
- Negative Lookahead

  `(?!pattern)`

- Negative Lookbehind

  `(?<!pattern)`

Negative Lookahead and Behind Examples

In [ ]:
```
open(my $fh, 'data/airports.tsv');

while (my $row = <$fh>) {
  chomp $row; #Remove trailing \n
  if ($row =~ /^.*\t.*International (?!Airport).*\t/){
      say $&;
  }
}
close($fh);
```

Splitting Strings

- Regular Expressions allow strings to be split in more dynamic ways

In [ ]:
```
$bad_csv_data = "Name,Phone Number,Email,a,list,of,websites,visited,Date";
@data = split /,(?=[A-Z])/, $bad_csv_data;
foreach $d (@data){
        if ($d =~ /,/ ){
            foreach $e (split /,/, $d, 2)
            {say $e}
        }
        else{
        say $d;
    }
}
```