Regular Expressions

Regular Expressions

- Regular expression describes a pattern
- `/pattern/` built-in datatype
  - Not a `String` object (unlike Java)
- Use `=~` to see if the pattern matches a string
  - Ambidextrous
- To add case-insensitivity specify an `i` flag
  - `/pattern/i`

Regular Expressions (Continued)

```
a = "Pretty cool stuff abc123"
puts "neat" if a =~ /cool/ # => neat
puts /abc/ =~ a # => 18 (position where the match occurs)
puts a =~ /123/ # => 21
p a =~ /jhu/ # => nil (puts would print out an empty string)

b = "cOOl stuff"
pattern1 = /cool/
pattern2 = /cool/i # case-insensitive
p b =~ pattern1 #=> nil
puts b =~ pattern2 # => 0
```

Regular Expressions (Continued)

- Some chars are special
  - `.` – matches any (1) character
  - Adjust the number of matched characters
    - `?` – match 0 or 1 of preceding pattern
    - `+` – match 1 or more of preceding pattern
    - `*` – match 0 or more of preceding pattern
    - `{number_of_times}`, `{min, }`, `{min, max}`, `{, max}`
  - Use backslash (`\`) to escape a special character

Regular expressions (Continued)

- Set `[ ]`
  - Specify a group of characters
  - `/[abc123]/ # ONE of these`
- Range `[x-y]`
  - Specify a group of characters via shortcut
  - `/[a-c1-3]/ # ONE in this range`
- Specify a `|` for an alternative

Regular Expressions (Continued)

- Special character classes
  - `\w` Any letter, digit, or underscore
  - `\W` Anything that `\w` doesn't match
  - `\d` Any digit
  - `\D` Anything that `\d` doesn't match (non-digits)
  - `\s` Whitespace (spaces, tabs, newlines etc.)
  - `\S` Non-whitespace (opposite of `\s`)

Regular Expressions (Continued)

```
p "ruby_class@gmail.com" =~ /.*@\w*/ # => 0
p "nice book" =~ /o{2}/ # => 6

html_element_pattern = /<\/?.*>/ # have to escape '/' since it is
                                 # also used to delimit the regular
                                 # expression itself
p html_element_pattern =~ "<p>" # => 0
p html_element_pattern =~ "</p>" # => 0

dr_jones = "Dr. Jones"
mr_jones = "Mr. Jones"
pattern = /.r\./
puts "will match both" if dr_jones =~ pattern and mr_jones =~ pattern
   # => will match both
```

Regular Expressions (Continued)

```
this_or_that_pattern = /this|that/
p this_or_that_pattern =~ "this" # => 0
p "that" =~ this_or_that_pattern # => 0

title = "Mr. Smith"
important_pattern = /[dm]r/i
puts "Someone important!" if title =~ important_pattern

def matches_social(str_to_check)
  social_sec_pattern = /[0-9]{3}-\d{2}-\d{2}/
  social_sec_pattern =~ str_to_check
end
p matches_social("1234567") # => nil
p matches_social("123-45-67") # => 0
```

Regular Expressions - Extraction

- Matching and Extraction
  - Sometimes it is useful to extract some text
  - Can be done by using `()` and `match`
  - Resulting array of the match returned
    - `arr[0]` – the full string that matched
    - `arr[1..n]` – the strings that were surrounded by parentheses

Regular Expressions (Continued)

```
address = "1000 Test Road, Columbia MD 21211"
city_state_pattern = /,\s*(\w*)\s*(\w*)\s/

# pull out city and state
bogus_result = "bogus address".match(city_state_pattern)
p bogus_result # => nil

good_result = address.match(city_state_pattern)
p good_result[0] # => , Columbia MD
city = good_result[1]
state = good_result[2]
puts "You live in #{city}, which is located in #{state}"
# => You live in Columbia, which is located in MD
```

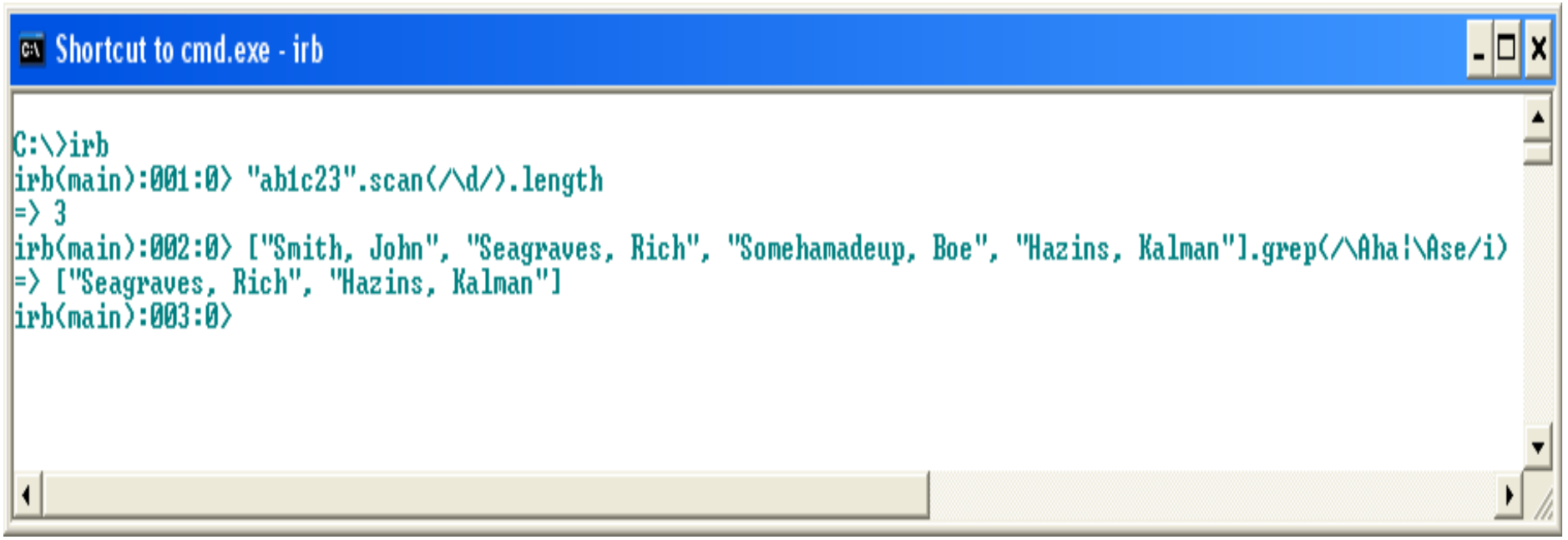
Extraction Perl style with $1..$n

```
address = "1000 Test Road, Columbia MD 21211"
city_state_pattern = /,\s*(\w*)\s*(\w*)\s/

# $1..$n matches the groupings inside ()

# pull out city and state
address =~ city_state_pattern
p $1 # => Columbia
p $2 # => MD
city = $1
state = $2
puts "You live in #{city}, which is located in #{state}"
# => You live in Columbia, which is located in MD
```

...Java...

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class HowManyDigits {
  public static void main(String[] args) {
    int numberOfDigits = 0;

    Pattern pat = Pattern.compile("\\d");
    Matcher matcher = pat.matcher("ab1c23");

    while (matcher.find()) {
      numberOfDigits++;
    }

    System.out.println("Total digits: " + numberOfDigits);
  }
}
```

…Ruby…


Shortcut to cmd.exe - irb
C:\>irb
irb(main):001:0> "ab1c23".scan(/\d/).length
=> 3
irb(main):002:0> ["Smith, John", "Seagraves, Rich", "Somehamadeup, Boe", "Hazins, Kalman"].grep(/\Aha!\Ase/i)
=> ["Seagraves, Rich", "Hazins, Kalman"]
irb(main):003:0>

Regular Expressions - Help

Check out rubular.com for help!