Problem Solving

- We can also use regexes to verify that the format provided as input matches what we expect
  - Example: Input string is in "DD/MM/YYYY" or "MM/DD/YYYY" format
  - Example: String provided is a URI (i.e., proper sets of characters)
- **Problem 3: Obtain a temperature (assumed to be Celsius) and return the number in Fahrenheit**
  - Number is to be an integer
  - There must be only one number in the string
  - No other characters (such as "C") should be at the end
  - Fahrenheit = (Celsius * 9 / 5) + 32

Input validation

```
#!/usr/bin/python

import re

print "Enter a temperature in Celsius:"
celsius = raw_input("> ")
celsius = celsius.rstrip("\n")


matchobj = re.search("^[0-9]+$", celsius)  # same as .match("\d+$",...)
if matchobj:
    celsius = int(celsius)
    fahrenheit = (celsius * 9 / 5) + 32
    print "%d C is %d F" % (celsius, fahrenheit)
else:
    print "Expecting a number, so I don't understand", celsius
```

```
$ ./prob03.py
Enter a temperature in Celsius:
> 30
30 C is 86 F
```

Problem Solving

- However, we should do a bit more
  - The problem statement is perhaps a bit too restrictive.
  - Negative temperatures cannot be given as values.
  - Decimal temperatures also cannot be provided
- The regular expression should accept these
  - And the other code changed to suit (i.e., use "float()" instead of "int()")

Input validation

```
#!/usr/bin/python

import re

print "Enter a temperature in Celsius:"
celsius = raw_input("> ")
celsius.rstrip("\n")


matchobj = re.search(r"^([-+]?[0-9]+(\.[0-9]*)?)$", input)
if matchobj:
    celsius, _ = matchobj.groups()
    celsius    = float(celsius)

    fahrenheit = (celsius * 9 / 5) + 32
    print "%.2f C is %.2f F" % (celsius, fahrenheit)
else:
    print "Expecting a number, so I don't understand", celsius
```

```
 ./prob03.py
Enter a temperature in Celsius:
> 12.2
12.20 C is 53.96 F
```

Problem Solving

- Our little script could be even more general
  - Rather than just convert from celsius to fahrenheit, it could convert the other direction
  - The starting value can be indicated by a "C" or "F" (or "c" or "f")
- **Problem 4: Obtain a temperature. If it is in celsius, return the number in fahrenheit; if in fahrenheit, return the number in celsius.**
  - Number can be an integer or a float, positive or negative
  - There must be only one number in the string
  - Character "C" or "F" implies what we are converting from and to.

Input validation plus more

```
#!/usr/bin/python

import re

print "Enter a temperature:"
celsius = raw_input("> ")
celsius.rstrip("\n")

matchobj = re.search(r"^([-+]?[0-9]+(\.[0-9]*)?)\s*([CF])$", input, re.IGNORECASE)
if matchobj:
    input_num, _, type = matchobj.groups()
    input_num           = float(input_num)

    if type == "C" or type == "c":
        celsius    = input_num
        fahrenheit = (celsius * 9 / 5) + 32
    else:
        fahrenheit = input_num
        celsius = (fahrenheit - 32) * 5 / 9

    print "%.2f C is %.2f F\n" % (celsius, fahrenheit)
else:
    print 'Expecting a number followed by "C" or "F",'
    print 'so I cannot interpret the meaning of', input
```

Notice how we indicate that case is to be ignored. The "re" module contains are large number of these kinds of options.

Problem Solving

- Our solution to Problem 4 still has some flaws
  - Cannot enter a number less than one without a leading zero.
  - No leading spaces are permitted (i.e., we have general whitespace issues)
  - We are using [0-9] instead of \d
  - etc. etc.
- There are many ways to "skin" a regular expression
  - The lesson so far, however, is that coming up with a full regular expression for these kinds of matches can be an iterative process.
  - Must also be aware of how a language deals with metasymbols within strings (e.g., Perl and Ruby are a bit different than Python)

Problem Solving

- A large set of programming problems with strings can be solved with substitutions
  - The pattern describes what we want to replace
  - Another string describes how we want it changed.
- There are a variety of substitution routines in the Python re module
  - We are interested in the one named re.sub()
  - It takes at least three parameters: search pattern, replacement pattern, and target string
- **Problem 5: Cleaning up stock prices**
  - Numbers arrive as strings from some stock-price service
  - Sometimes they have lots of trailing zeros
  - We want to take the first two digits after the decimal point, and take the third digit only if is not zero; all other digits are removed
  - Example: "3.14150002" --> "3.141"
  - Example: "51.5000" --> "51.50"

First, a warm up

```
#!/usr/bin/python

import re

line1 = "Michael Zastre"
line2 = "Michael Marcus Joseph Zastre"

print "Before:", line1
line1 = re.sub("Michael", "Mike", line1)
print "After:", line1

print

print "Before:", line2
line2 = re.sub("Marcus", "M.", line2)
line2 = re.sub("Joseph", "J.", line2)
print "After:", line2
```

Substitutions are global (i.e., all instances for a particular string match get substituted).

```
$ ./warmup.py
Before: Michael Zastre
After: Mike Zastre

Before: Michael Marcus Joseph Zastre
After: Michael M. J. Zastre
```

Problem Solving

- **Problem 5: Cleaning up stock prices**
  - Numbers arrive strings from some stock-price service
  - Sometimes they have lots of trailing zeros
  - We want to take the first two digits after the decimal point, and take the third digit only if is not zero; all other digits are removed
  - Example: "3.14150002" --> "3.141"
  - Example: "51.5000" --> "51.50"
- Let's think this through:
  - We are not interested in changing digits to the left of the decimal point.
  - We want at least two digits to the right of the decimal point.
  - If the third digit to the right of the decimal point is not a zero, then we want to keep it...
  - ... otherwise we don't want it.
- We'll throw into the mix one other feature
  - Match references (i.e., \<num>)

Substitutions

```
#!/usr/bin/python

import re

prices =[ "3.141500002", "12.125", "51.500"]

for p in prices:
    print "Before --> ", p
    p = re.sub(r"(\.\d\d[1-9]?)\d*", r"\1", p)
    print "After --> ", p
    print
```

In the second parameter to re.sub(), all backslash escapes are processed (i.e. Python string rules), so we need to use r" " to denote the string with the backreference.

```
$ ./prob05.py
Before -->  3.141500002
After -->  3.141

Before -->  12.125
After -->  12.125

Before -->  51.500
After -->  51.50
```

Problem Solving

- Python supports **shortstrings** and **longstrings**
  - All of our strings so far have been of the short form
  - Docstrings are longstrings (strings delimited with """)
  - We can use longstrings to format a textual document
- **Problem 6: Nigerian ~~Spam~~ Form Letters**
  - (Please don't do this at home.)
  - We have a text block that we want to customize
  - There are certain spots in the text block where we have "tags" that must be replaced with specific strings
  - We would like to do this with regular expressions

Example: Form letter

```
=LOCATION=

Attention: =TITLE=

Having consulted with my colleagues and based on the
information gathered from the Nigerian Chambers of Commerce
and industry, I have the privilege to request for your
assistance to transfer the sum of =AMOUNT=
(=AMOUNTSPELLED=) into your accounts.

We are now ready to transfer =AMOUNT= and that is where
you, =SUCKER=, come in.
```

```
place = 'Lagos, Nigeria'
title = 'The President/CEO'
cash  = '$47,500,000.00'
cashtext = 'forty-seven million, five hundred thousand dollars'
important_person = 'Mr. Abercrombie'
```

Form Letter

- To fill out the form letter, we could have the following subtitutions:
  - contents of "place" replace all spots with "=LOCATION="
  - contents of "title" replace all spots with "=TITLE="
  - contents of "cash" replace all spots with "=AMOUNT="
  - contents of "cashtext" replace all spots with "=AMOUNTSPELLED="
  - contents of "important_person" replace all spots with "=SUCKER="
- This can be implemented via a straight-forward sequence of re.sub() operations
  - By default, the operation performs a global replacement on the target string
  - (However, we can use re.subn() if we want to limit this.)

Form letter

```
#!/usr/bin/python

import re

letter = """
    =LOCATION=

    Attention: =TITLE=

    Having consulted with my colleagues and based on the
    information gathered from the Nigerian Chambers of Commerce
    and industry, I have the privilege to request for your
    assistance to transfer the sum of =AMOUNT=
    (=AMOUNTSPELLED=) into your accounts.

    We are now ready to transfer =AMOUNT= and that is where
    you, =SUCKER=, come in."""

# continued on next slide
```

Form letter

```
# continued from previous slide

place = 'Lagos, Nigeria'
title = 'The President/CEO'
cash  = '$47,500,000.00'
cashtext = 'forty-seven million, five hundred thousand dollars'
important_person = 'Mr. Abercrombie'

letter = re.sub(r"=LOCATION=", place, letter)
letter = re.sub(r"=TITLE=", title, letter)
letter = re.sub(r"=AMOUNT=", cash, letter)
letter = re.sub(r"=AMOUNTSPELLED=", cashtext, letter)
letter = re.sub(r"=SUCKER=", important_person, letter)

print letter
```

Example: Form letter

```
Lagos, Nigeria

   Attention: The President/CEO

   Having consulted with my colleagues and based on the
   information gathered from the Nigerian Chambers of Commerce
   and industry, I have the privilege to request for your
   assistance to transfer the sum of $47,500,000.00
   (forty-seven million, five hundred thousand dollars) into your accounts.

   We are now ready to transfer $47,500,000.00 and that is where
   you, Mr. Abercrombie, come in.
```