Introduction to Programming for Scientists

Prof. Steven Ludtke
N420, sludtke@bcm.edu

Team Learning 9
Regular Expressions

The goal

Become familiar with Regular Expressions.

Whenever there is a question, fill in your answer in the Jupyter notebook!

re functions (20 min)

- Start with this:

```
import re

string1="This is another string"
string2="This is my test string, with several Tests in testing,
like tust"

print(string1.find("test"))
print(string2.find("test"))
print(re.search("test",string1))
print(re.search("test",string2))
print(re.search("test",string2).span())
```

- Look at each of the 5 lines of output, and make sure you understand why you got each one. Understand the difference between the output from find() and re.search()
- Now, let's try match()

```
print(re.match("test",string2))
print(re.match(".*test",string1))
print(re.match(".*test",string2))
```

- Make sure you understand each of these lines, in particular, why the first test returned None. ".*" is the regex equivalent of the wildcard "*", that is, any string of unlimited length.
- A key feature of regular expressions over string.find() is the ability to find ALL of the matching substrings. Try:

```
print(re.findall("test",string2))
print(re.findall("(?i)test",string2))
```

- The second version matches "test" or "Test". Find another regular expression which matches both "test" and "Test", and try it to make sure it works. If you can't figure it out continue through the exercise and come back to this later.
- The problem with findall() is that it returns only the matching substrings. It doesn't tell you where the substrings are. This can be enough in some cases, but most of the time we'd like the full information:

```
for m in re.finditer("(?i)test",string2):
     print(m)
```

- The Match objects returned by finditer() have more information internally than you see on the screen when you just print the match. You can look them up in the "re" documentation to find more.
- Next, split():

```
print(re.split("test",string2))
```

- How is this different from string2.split("test")? Could you do this with string.split() ?

```
print(re.split("(?i)test",string2))
```

- Finally, sub(), which is similar to string.replace:

```
print(re.sub("(?i)test","TEST",string2))
```

Building Regular Expressions - core concepts (20 min)

- The core of a regular expression is normal text. As we saw above the regular expression "test" matches exactly that, the letters t, e, s and t in that exact order. We then build upon that base with various extensions:
- [ ] - one of several choices for a single letter. Keep in mind that regular expressions are an entirely different language from Python. Square braces do not have the same meaning.

```
print(re.findall("[Tt]est",string2))
```

- '.' - a period matches any single character.

```
print(re.findall("t.st",string2))
```

- '\' - escape. If you want to search a string for a special character, like "." you must protect it with the escape character, eg "\."
- '*' - unlimited copies of the previous element. For example:

```
print(re.findall("t.*st",string2))
```

- Did this do what you expected? By default, repetition operators like '*', '+' and '?' are "greedy". That is, they will match as much as they possibly can. You can use the '?' suffix to make them non-greedy:

```
print(re.findall("t.*?st",string2))
```

- This can still lead to some surprises, though, and you may begin to understand why crafting bug-free regular expressions can be very tricky.
- '\s' - matches any whitespace character
- | - vertical bar means "or". "a|b" matches "a" or "b", but applies to whole regular expressions:

```
print(re.findall("Test|test",string2))
```

- We have only scratched the surface here, but this gives you some of the basic capabilities. See: https://docs.python.org/2/library/re.html for more.

Exercises (the rest of class)

Below are a set of practice exercises. Do as many as you can before the end of class, and send in your notebooks. You are not required to continue doing these after class ends:

- Recall the earlier lecture where we had to take a genetic sequence which included numbers, spaces, etc. and remove all of the extraneous stuff, to leave behind just the sequence. See if you can use regular expressions to turn this string into pure sequence information:

```
seq="""
0000\tGAAAAATACT TACTAAGGCG TTTTTTATTT GGTGATATTT
0040\tTTTTCAATAT CATGCAGCAA ACGGTGCAAC ATTGCCGTGT
0080\tCTCGTTGCTC TAAAAGCCCC AGGCGTTGTT GTAACCAGTC
0120\tGACCAGTTTT ATGTCATCTG CCACTGCCAG AGTCGTCAGC"""
```

- When storing multicolumn numerical data in a file, different separators are used at different times. Write a function myfn(instring) which will return a list of float() values for any of the following (ie - the same function should work with any of these):

```
str1="1\t3\t5\t9.0"
str2="1.0 3.0 5 9.0"
str3="1, 3, 5, 9"
```

You should be able to call myfn() on any of these strings and get back [1.0,3.0,5.0,9.0]

- There are two stages to writing a program to interpret a mathematical expression, like "5+5*10/7". The first step is to separate the operators from the numbers. Write a function which can take the following string and return a list of numbers and a list of binary operators (like "+", "-", etc.). You should always have 1 more number than operator:

```
str="100.0*50+27/222-5"
```

- Extra challenge, make the function work properly for this:

```
str="100.0*-50+27/222-5"
```