“Everything Else”

Find all substrings

We've learned how to find the first location of a string in another string with find. What about finding all matches?

Start by looking at the documentation.

```
S.find(sub [,start [,end]]) -> int

Return the lowest index in S where substring sub is found,
such that sub is contained within s[start,end].  Optional
arguments start and end are interpreted as in slice notation.

Return -1 on failure.
```

Experiment with find

```
>>> seq = "aaaaTaaaTaaT"
>>> seq.find("T")
4
>>> seq.find("T", 4)
4
>>> seq.find("T", 5)
8
>>> seq.find("T", 9)
11
>>> seq.find("T", 12)
-1
>>>
```

How to program it?

The only loop we've done so far is "for".
But we aren't looking at every element in the list.
We need some way to jump forward and stop when done.

`while` statement

The solution is the *while* statment

```
>>> pos = seq.find("T")
>>> while pos != -1:
...     print "T at index", pos
...     pos = seq.find("T", pos+1)
...
T at index 4
T at index 8
T at index 11
>>>
```

While the test is true

Do its code block

There's duplication...

Duplication is bad. (Unless you're a gene?)
The more copies there are the more likely some will be different than others.

```
>>> pos = seq.find("T")
>>> while pos != -1:
...     print "T at index", pos
...     pos = seq.find("T", pos+1)
...
T at index 4
T at index 8
T at index 11
>>>
```

The `break` statement

The break statement says “exit this loop immediately” instead of waiting for the normal exit.

```
>>> pos = -1
>>> while 1:
...     pos = seq.find("T", pos+1)
...     if pos == -1:
...         break
...     print "T at index", pos
...
T at index 4
T at index 8
T at index 11
>>>
```

`break` in a `for`

A `break` also works in the `for` loop

Find the first 10 sequences in a file which have a poly-A tail

```
sequences = []
for line in open(filename):
    seq = line.rstrip()
    if seq.endswith("AAAAAAAA"):
        sequences.append(seq)
    if len(sequences) > 10:
        break
```

elif

Sometimes the if statement is more complex than if/else

"If the weather is hot then go to the beach. If it is rainy, go to the movies. If it is cold, read a book. Otherwise watch television."

```
if is_hot(weather):
    go_to_beach()
elif is_rainy(weather):
    go_to_movies()
elif is_cold(weather):
    read_book()
else:
    watch_television()
```

tuples

Python has another fundamental data type - a *tuple*.
A tuple is like a list except it's immutable (can't be changed)

```
>>> data = ("Cape Town", 2004, [])
>>> print data
('Cape Town', 2004, [])
>>> data[0]
'Cape Town'
>>> data[0] = "Johannesburg"
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: object doesn't support item assignment
>>> data[1:]
(2004, [])
>>>
```

Why tuples?

We already have a list type. What does a tuple add?

This is one of those deep computer science answers.

Tuples can be used as dictionary keys, because they are immutable so the hash value doesn't change.

Tuples are used as anonymous classes and may contain heterogeneous elements. Lists should be homogenous (eg, all strings or all numbers or all sequences or...)

String Formating

So far all the output examples used the print statement. Print puts spaces between fields, and sticks a newline at the end. Often you'll need to be more precise.

Python has a new definition for the "%" operator when used with a strings on the left-hand side - "string interpolation"

```
>>> name = "Andrew"
>>> print "%s, come here" % name
Andrew, come here
>>>
```

Simple string interpolation

The left side of a string interpolation is always a string. The right side of the string interpolation may be a dictionary, a tuple, or anything else. Let's start with the last.

The string interpolation looks for a "%" followed by a single character (except that "%%" means to use a single "%"). That letter immediately following says how to interpret the object; %s for string, %d for number, %f for float, and a few others

Most of the time you'll just use %s.

% examples

Also note some of the special formating codes.

```
>>> "This is a string: %s" % "Yes, it is"
'This is a string: Yes, it is'
>>> "This is an integer: %d" % 10
'This is an integer: 10'
>>> "This is an integer: %4d" % 10
'This is an integer:   10'
>>> "This is an integer: %04d" % 10
'This is an integer: 0010'
>>> "This is a float: %f" % 9.8
'This is a float: 9.800000'
>>> "This is a float: %.2f" % 9.8
'This is a float: 9.80'
>>>
```

string % tuple

To convert multiple values, use a tuple on the right.
(Tuple because it can be heterogeneous)
Objects are extracted left to right. First % gets the first element in the tuple, second % gets the second, etc.

```
>>> "Name: %s, age: %d, language: %s" % ("Andrew", 33, "Python")
'Name: Andrew, age: 33, language: Python'
>>>
```

The number of % fields and tuple length must match.

```
>>> "Name: %s, age: %d, language: %s" % ("Andrew", 33)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: not enough arguments for format string
>>>
```

string % dictionary

When the right side is a dictionary, the left side must include a name, which is used as the key.

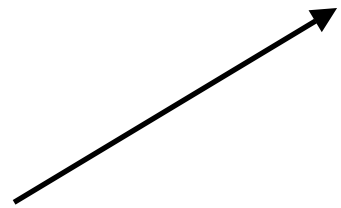
```
>>> d = {"name": "Andrew",
...       "age": 33,
...       "language": "Python"}
>>>
>>> print "%(name)s is %(age)s years old.  Yes, %(age)s." % d
Andrew is 33 years old.  Yes, 33.
>>>
```

A %(names)s may be duplicated and the dictionary size and % count don't need to match.

Writing files

Opening a file for writing is very similar to opening one for reading.

```
>>> infile = open("sequences.seq")
>>> outfile = open("sequences_small.seq", "w")
```



Open file for writing

The `write` method

```
>>> infile = open("sequences.seq")
>>> outfile = open("sequences_small.seq", "w")
>>> for line in infile:
...     seq = line.rstrip()
...     if len(seq) < 1000:
...         outfile.write(seq)
...         outfile.write("\n")
...
>>> outfile.close()
>>> infile.close()
>>>
```

I need to write my own newline.

The close is optional, but good style. Don't fret too much about it.

Command-line arguments

I mentioned this in the advanced exercises for Thursday. See there for full details.

The short version is that Python gives you access to the list of Unix command-line arguments through sys.argv, which is a normal Python list.

```
% cat show_args.py
import sys
print sys.argv
% python show_args.py
['show_args.py']
% python show_args.py 2 3
['show_args.py', '2', '3']
% python show_args.py "Hello, World"
['show_args.py', 'Hello, World']
%
```

Exercise 1

The hydrophobic residues are [FILAPVM].

Write a program which asks for a protein sequence and prints "Hydrophobic signal" if (and only if) it has at least 5 hydrophobic residues in a row. Otherwise print "No hydrophobic signal."

Some test cases are listed on the next page.

Test cases for #1

Protein sequence? FILAEPVM
No hydrophobic signal

Protein sequence? FILA
No hydrophobic signal

Protein sequence? QQPLIMAW
Hydrophobic signal

Protein sequence? AA
No hydrophobic signal

Protein sequence? AAAAAAAAAA
Hydrophobic signal

Protein sequence? AAFILAPILA
Hydrophobic signal

Protein sequence? ANDREWDALKE
No hydrophobic signal

Exercise #2

Modify your solution from Exercise #1 so that it prints "Strong hydrophobic signal" if the input sequence has 7 or more hydrophobic residues in a row, print "Weak hydrophobic signal" if it has 3 or more in a row. Otherwise, print "No hydrophobic signal."

Some test cases

Protein sequence? FILAEPVM
Weak hydrophobic signal

Protein sequence? FILA
Weak hydrophobic signal

Protein sequence? QQPLIMAW
Weak hydrophobic signal

Protein sequence? AA
No hydrophobic signal

Protein sequence? AAAAAAAAAA
Strong hydrophobic signal

Protein sequence? AAFILAPILA
Strong hydrophobic signal

Protein sequence? ANDREWDALKE
No hydrophobic signal

Exercise #3

The Prosite pattern for a Zinc finger C2H2 type domain signature is

```
C.{2,4}C.{3}[LIVMFYWC].{8}H.{3,5}
```

Based on the pattern, create a sequence which is matched by it. Use Python to test that the pattern matches your sequence.

Exercise #4 (hard)

The (made-up) enzyme APD1 cleaves DNA. It recognizes the sequence GAATTC and separates the two thymines. Every such site is cut so if that pattern is present N times then the fully digested result has N+1 sequences.

Write a program to get a DNA sequence from the user and "digest" it with APD1. For output print each new sequence, one per line. Hint: Start by finding the location of all cuts.

See the next page for test cases.

Test cases for #4

```
Enter DNA sequence: A
A


Enter DNA sequence: GAATTC
GAAT
TC


Enter DNA sequence: AGAATTCCCAAGAATTCCTTTGAATTCAGTC
AGAAT
TCCCAAGAAT
TCCTTTGAAT
TCAGTC
```