Regular Expressions

- Regular expressions (regexp) are a text-matching tool embedded in Python
- They are useful in creating string searches and string modifications
- You can always use regular Python instead, but regexps are often much easier
- Documentation: `http://docs.python.org/library/re.html`

Basics of regexp construction

- Letters and numbers match themselves
- Normally case sensitive
- Watch out for punctuation–most of it has special meanings!

Matching one of several alternatives

- Square brackets mean that any of the listed characters will do
- `[ab]` means either "a" or "b"
- You can also give a range:
- `[a-d]` means "a" "b" "c" or "d"
- Negation: caret means "not"

```
[^a-d]     # anything but a, b, c or d
```

Wild cards

- "." means "any character"
- If you really mean "." you must use a backslash
- WARNING:
  - backslash is special in Python strings
  - It's special again in regexps
  - This means you need too many backslashes
  - We will use "raw strings" instead
  - Raw strings look like `r"ATCGGC"`

Using . and backslash

- To match file names like "hw3.pdf" and "hw5.txt":

```
hw.\....
```

Zero or more copies

- The asterisk repeats the previous character 0 or more times
- "ca*t" matches "ct", "cat", "caat", "caaat" etc.
- The plus sign repeats the previous character 1 or more times
- "ca+t" matches "cat", "caat" etc. but not "ct"

Repeats

- Braces are a more detailed way to indicate repeats
- A{1,3} means at least one and no more than three A's
- A{4,4} means exactly four A's

Practice problem 1

- Write a regexp that will match any string that starts with "hum" and ends with "001" with any number of characters, including none, in between

- (Hint: consider both "." and "*")

Practice problem 2

- Write a regexp that will match any Python (.py) file.
- There must be at least one character before the ".".
- ".py" is not a legal Python file name
- (Imagine the problems if you imported it!)

Using the regexp

First, compile it:

```
import re
myrule = re.compile(r".+\.py")
print myrule
<_sre.SRE_Pattern object at 0xb7e3e5c0>
```

The result of `compile` is a Pattern object which represents your regexp

Using the regexp

Next, use it:

```
mymatch = myrule.search(myDNA)
print mymatch
None
mymatch = myrule.search(someotherDNA)
print mymatch
<_sre.SRE_Match object at 0xb7df9170>
```

The result of `match` is a Match object which represents the result.

All of these objects! What can they do?

Functions offered by a Pattern object:

- `match()`–does it match the beginning of my string? Returns None or a match object
- `search()`–does it match anywhere in my string? Returns None or a match object
- `findall()`–does it match anywhere in my string? Returns a list of strings (or an empty list)
- Note that `findall()` does NOT return a Match object!

All of these objects! What can they do?

Functions offered by a Match object:

- `group()`–return the string that matched
  `group()`–the whole string
  `group(1)`–the substring matching 1st parenthesized sub-pattern
  `group(1,3)`–tuple of substrings matching 1st and 3rd parenthesized sub-patterns

- `start()`–return the starting position of the match

- `end()`–return the ending position of the match

- `span()`–return (start,end) as a tuple

Regular expressions summary

- The `re` module lets us use regular expressions
- These are fast ways to search for complicated strings
- They are not essential to using Python, but are very useful
- File format conversion uses them a lot
- Compiling a regexp produces a Pattern object which can then be used to search
- Searching produces a Match object which can then be asked for information about the match