

Solutions, Week 2, Day 4

1. Write an input function based on `raw_input` that will ask the user for an integer and - if the the input is not correct and one is not given - repeats its request.

```
def askForInt (prompt="Enter an integer"):
    theAnswer = ""
    import re
    theAnswerPattern = "^\\d+$"
    theAnswerRe = re.compile (theAnswerPattern)
    while (re.search (theAnswer) == None):
        theAnswer = raw_input (prompt)
    return theAnswer
```

2. Write a program that takes a name of a raw sequence file as an argument. It will process this file looking for sequencing errors or problems. These are defined as more than two unknowns ? in a row. Save the sequence with these regions replaced by three asterisks. `input2-4a.seq` can be regarded as a typical input.

```
def replaceErrors (filename):
    theInFile = open (filename, 'r')
    theRawSeq = theInFile.readline()
    theInFile.close()

    theRawSeq = theRawSeq.strip()
    import re
    theRe = re.compile (r'\?{3,}')
    theNewSeq = theRe.sub (theRawSeq, "***")

    theOutFile = open (filename, 'w')
    theOutFile.write (theNewSeq)
    theOutFile.close()

if __name__ == '__main__':
    import sys
    replaceErrors (sys.argv[1])
```

3. Modify it so it can accept sequences that are spread over several lines and formatted as in `input2-4b.seq`. Note however that the output does not have to be formatted.

```
def replaceErrors (filename):
    theInFile = open (filename, 'r')
    theRawSeqList = theInFile.readlines()
    theInFile.close()

    theRawSeq = ''.join (theRawSeqList)
    import re
```

```

theRe = re.compile (r'\s+')
theRawSeq = theRe.sub (theRawSeq, "")

theRawSeq = theRawSeq.strip()
theRe = re.compile (r'\s{3,}')
theNewSeq = theRe.sub (theRawSeq, "***")

theOutFile = open (filename, 'w')
theOutFile.write (theNewSeq)
theOutFile.close()

```

4. Modify the program so it replaces the error regions with the exact number of asterisks.

```

theRe = re.compile (r'\s{3,}')
theMatch = theRe.search (theRawSeq)
while (theMatch):
    theHit = theMatch.group()
    theRep = "*" * len (theHit)
    theRawSeq = theRawSeq.replace (theHit, theRep)
    theMatch = theRe.search (theRawSeq)

```

5. Write a parser for a phylip file. Descriptions can be found [here](#), [here](#) and [here](#). An example is given as `input2-4c.phylip` and you can assume this is typical. The contents should be returned as something logical (e.g. a list of name-sequence pairs). You can assume there are no errors in the file. Modules and functions like regular expressions, `split` and `replace` will be very useful.

```

def parsePhylip (infile):
    theCurrLine = infile.readline()
    theResults = []
    while theCurrLine.startswith(">"):
        startsymbol, header = theCurrLine.split(num=1)
        theSeq = ""
        theCurrLine = infile.readline()
        while (not theCurrLine.startswith(">")) and theCurrLine:
            theSeq = theSeq + theCurrLine
            theResults.append ((header, theSeq))
    return theResults

```