Inf2a: Lab 2 (Solutions)

Object-Oriented Programming and NLTK

EXCERCISE 1

```
class Queue:
    def __init__(self):
        self.q = []

    def isempty(self):
        return len(self.q)==0

    def push(self,item):
        self.q.append(item)

    def pop(self):
        if not self.isempty():
            return self.q.pop(0)
```

EXCERCISE 2

```
class Stack:
    def __init__(self):
        self.s = []

    def isempty(self):
        return len(self.s)==0

    def push(self,item):
        self.s.append(item)

    def pop(self):
        if not self.isempty():
            return self.s.pop()
```

EXCERCISE 3

```
class Checker:
    def checkPrefix(self, list, prefix):
        for i in list:
            if i[:2]==prefix: print '*', i[:2], i
            else: print ' ', i[:2], i

>>> import oo_checker
>>> c = oo_checker.Checker()
>>> c.checkPrefix(lst,'wh')
```

EXCERCISE 4

```
class Checker:
    def __init__(self,infix):
        self.infix = infix

    def check(self, str):
        return self.infix in str
```

EXCERCISE 5

```
class AddressBook:
    def __init__(self):
        self.b = {}

    def insert(self,name, phone):
        self.b[name]=phone

    def get(self,name):
        return self.b[name]

    def has_name(self,name):
        return self.b.has_key(name)

    def list(self):
        for n,p in self.b.iteritems():
            print n,p

    def delete(self, name):
        del self.b[name]

    def orderedList(self):
        orderedkeys = self.b.keys()
        orderedkeys.sort()
        for n in orderedkeys:
            print n, self.b[n]
```

EXCERCISE 6

```
>>> from nltk.tokenize import RegexpTokenizer
>>> text = 'The interest does not exceed 8.25%.'
>>> pattern = r'\d+\.\d+\%|\w+|\$\d+\.\d+|[^\w\s]+'
>>> tokenizer = RegexpTokenizer(pattern)
>>> tokenizer.tokenize(text)
['The', 'interest', 'does', 'not', 'exceed', '8.25%', '.']
```

EXCERCISE 7

```
>>> from nltk.corpus import brown
>>> from nltk.probability import ConditionalFreqDist
>>> cfdist = ConditionalFreqDist()
>>> prev=None
>>> for fileid in brown.fileids():
...     for sntc in brown.sents(fileid):
...         for token in sntc:
...             cfdist[prev][token] += 1
...             prev=token
...
>>> word='an'
>>> import random
>>> for i in range(20):
...     print word,
...     lwords=cfdist[word].keys()
...     if (len(lwords) > 10):
...         lwords = lwords[:10]
...     word= random.choice(lwords)
...     while word == '.':
...         word = random.choice(lwords)
...
```

EXCERCISE 8

Parse the sentence Mary saw the dog with the telescope. How many parse trees do you get?

2

For grammar3, write down 2 unambiguous sentences (just one tree), 2 ambiguous sentences (more than one tree), and 2 ungrammatical sentences (no tree at all).

unambiguous: “Mary saw the dog”” and “Mary ate the apple””

ambiguous: “I saw the dog on the telescope”” and “I saw an apple with the telescope””

ungramatical: "I see the dog with the telescope" and "Mary saw on the telescope the apple"

EXCERCISE 9

Extend the program to identify the subject in all the sentences in "wsj_0003":

```
>>> for t in treebank.parsed_sents('wsj_0003.mrg'):
...     for ch_tree in t:
...         if (ch_tree.label().startswith('NP-SBJ')):
...             print ch_tree.leaves()
```

Extend the code to identify all the subjects in a given sentence. A subordinate clause in a sentence will have its own subject. Using recursion, print all the subjects in the sentence:

```
>>> def printsub(t):
...     if (t.node.startswith('NP-SBJ')):
...         print t.leaves()
...     else:
...         if (t.height() > 2):
...             for i in t:
...                 printsub(i)
>>> c = 1
>>> for t in treebank.parsed_sents('wsj_0003.mrg'):
...     print c
...     printsub(t)
...     c += 1
```