Akmal Ataev

NYU Student ID aa44

HW4

EX 47

In [1]:

```
class Room(object):

    def __init__(self, name, description):
        self.name = name
        self.description = description
        self.paths = {}

    def go(self, direction):
        return self.paths.get(direction, None)

    def add_paths(self, paths):
        self.paths.update(paths)
```

In [2]:

```
def test_room():
    gold = Room("GoldRoom",
                """ This room has gold in it you can grab. There's a \
    door to the north."""
                )
    assert_equal(gold.name, "GoldRoom")
    assert_equal(gold.paths, {})

def test_room_paths():
    center = Room("Center", "Test room in the center.")
    north = Room("North", "Test room in the north.")
    south = Room("South", "Test room in the south.")

    center.add_paths({'north': north, 'south': south})
    assert_equal(center.go('north'), north)
    assert_equal(center.go('south'), south)

def test_map():
    start = Room("Start", "You can go west and down a hole.")
    west = Room("Trees", "There are trees here, you can go east.")
    down = Room("Dungeon", "It's dark down here, you can go up.")

    start.add_paths({'west': west, 'down': down})
    west.add_paths({'east': start})
    down.add_paths({'up': start})

    assert_equal(start.go('west'), west)
    assert_equal(start.go('west').go('east'), start)
    assert_equal(start.go('down').go('up'), start)
```

HW 48

In [3]:

```
class Lexicon(object):

    def convert_number(s):
        try:
            return int(s)
        except ValueError:
            return None

    def scan(s):
        directions = ['north', 'south', 'west', 'east']
        verbs = ['go', 'kill', 'eat']
        stops = ['the', 'in', 'of']
        nouns = ['bear', 'princess']

        result = []

        words = s.lower().split()

        for word in words:
            if word in directions:
                result.append(('direction', word))
            elif word in verbs:
                result.append(('verb', word))
            elif word in stops:
                result.append(('stop', word))
            elif word in nouns:
                result.append(('noun', word))
            elif Lexicon.convert_number(word):
                result.append(('number', int(word)))
            else:
                result.append(('error', word))

        return result
```

In [4]:

```
def text_directions():
    assert_equal(lexicon.scan("north"), [('direction', 'north')])
    result = lexicon.scan("north south east")
    assert_equal(result, [('direction', 'north'),
                          ('direction', 'south'),
                          ('direction', 'east')
                          ])


def test_verbs():
    assert_equal(lexicon.scan("go"), [('verb', 'go')])
    result = lexicon.scan("go kill eat")
    assert_equal(result, [('verb', 'go'),
                          ('verb', 'kill'),
                          ('verb', 'eat')
                          ])


def test_stops():
    assert_equal(lexicon.scan("the"), [('stop', 'the')])
    result = lexicon.scan("the in of")
    assert_equal(result, [('stop', 'the'),
                          ('stop', 'in'),
                          ('stop', 'of')
                          ])


def test_nouns():
    assert_equal(lexicon.scan("bear"), [('noun', 'bear')])
    result = lexicon.scan("bear princess")
    assert_equal(result, [('noun', 'bear'),
                          ('noun', 'princess')])
```

```
def test_numbers():
    assert_equal(lexicon.scan("1234"), [('number', '1234')])
    result = lexicon.scan("3 91234 23098 8128 0")
    assert_equal(result, [('number', '3'),
                          ('number', '91234')
                          ])


def test_errors():
    assert_equal(lexicon.scan("ASDFADFASDF"), [('error', 'ASDFADFASDF')])
    result = lexicon.scan("Bear IAS princess")
    assert_equal(result, [('noun', 'Bear'),
                          ('error', 'IAS'),
                          ('noun', 'princess')
                          ])
```

HW 49

In [5]:

```
# parse.py
class ParserError(Exception):
    pass

class Sentence(object):

    def __init__(self, subject, verb, object):
        # remember we take ('noun','princess') tuples and convert them
        self.subject = subject[1]
        self.verb = verb[1]
        self.object = object[1]


def peek(word_list):

    if word_list:
        word = word_list[0]
        return word[0]
    else:
        return None


def match(word_list, expecting):
    if word_list:
        # notice the pop function here
        word = word_list.pop(0)
        if word[0] == expecting:
            return word
        else:
            return None
    else:
        return None


def skip(word_list, word_type):
    while peek(word_list) == word_type:
        # remove words that belongs to word_type from word_list
        match(word_list, word_type)


class Parser(object):

    def parse_verb(self, word_list):
        skip(word_list, 'stop')

        if peek(word_list) == 'verb':
            return match(word_list, 'verb')
        else:
            raise ParserError("Expected a verb next.")
```

```
    def parse_object(self, word_list):
        skip(word_list, 'stop')
        next = peek(word_list)

        if next == 'noun':
            return match(word_list, 'noun')
        if next == 'direction':
            return match(word_list, 'direction')
        else:
            raise ParserError("Expected a noun or direction next.")


    def parse_subject(self, word_list, subj):
        verb = self.parse_verb(word_list)
        obj = self.parse_object(word_list)

        return Sentence(subj, verb, obj)


    def parse_sentence(self, word_list):
        skip(word_list, 'stop')

        start = peek(word_list)

        if start == 'noun':
            subj = match(word_list, 'noun')
            return self.parse_subject(word_list, subj)
        elif start == 'verb':
            # assume the subject is the player then
            return self.parse_subject(word_list, ('noun', 'player'))
        else:
            raise ParserError("Must start with subject, object, or verb not: %s" % start
)
```

In [6]:

```
# from nose.tools import *
# from ex49.parser import *
# from ex48.lexicon import *
# from copy import deepcopy
# parser_tests.py file

# # construct a test set that consists of several test lists
# global_test_lists = [ scan('south'), scan('door'), scan('go'), scan('to'),
#                       scan('234'), scan('error123'), scan('the east door'), scan('go t
o east'),
#                       scan('bear go to the door'), scan('the princess kill 10 bears')
#                     ]

# the type of the the first tuple for each test list
# test_types = ['direction', 'noun', 'verb', 'stop', 'number', 'error',
#                 'stop', 'verb', 'noun', 'stop', None]

# list_len = len(global_test_lists)


# def test_peek():
#     ''' test peek function '''
#     test_lists = deepcopy(global_test_lists)
#     for i in range(list_len):
#         test_list = test_lists[i]
#         expected_word = test_types[i]
#         assert_equal(peek(test_list), expected_word)


# def test_match():
#     ''' test match function '''
#     test_lists = deepcopy(global_test_lists)
#     for i in range(list_len):
```

```
#         test_list = test_lists[i]
#         test_type = test_types[i]
#         if len(test_list) > 0:
#             expected_tuple = test_list[0]
#         else:
#             expected_tuple = None
#         assert_equal(match(test_list, test_type), expected_tuple)



# def test_skip():
#     ''' test skip function '''
#     test_lists = deepcopy(global_test_lists)
#     expected_lists1 = [scan('south'), scan('door'), scan('go'), [], scan('234'), scan('
error123'),
#                        scan('east door'), scan('go to east'), scan('bear go to the doo
r'),
#                        scan('princess kill 10 bear'), []]

#     for i in range(list_len):
#         test_list = test_lists[i]
#         expected_list = expected_lists1[i]
#         skip(test_list, 'stop')
#         assert_equal(test_list, expected_list)

#     test_list2 = [('error', 'error123')]
#     expected_list2 = []
#     skip(test_list2, 'error')
#     assert_equal(test_list2, expected_list2)



# def test_parse_verb():
#     ''' test parse_verb function '''
#     parser = Parser()
#     # test good situations
#     test_lists_good =  [scan('go'), scan('go to east'), scan('to error123 eat')]

#     expected_lists = [scan('go'), scan('go'), scan('eat')]

#     for i in range(len(test_lists_good)):
#         test_list = test_lists_good[i]
#         expected_list = expected_lists[i]
#         assert_equal(parser.parse_verb(test_list), *expected_list)

#     # test bad situations
#     test_lists_bad = [scan('south'), scan('door'), scan('234'), scan('east door'),
#                       scan('error123'), scan('to'),
#                      scan('bear go to the door'), scan('the princess kill 10 bear'),
[]]
#     for i in range(len(test_lists_bad)):
#         test_list = test_lists_bad[i]
#         assert_raises(ParserError, parser.parse_verb, test_list)


# def test_parse_num():
#     ''' test parse_num function '''
#     parser = Parser()
#     # test good situations
#     test_lists_good = [scan('302'), scan('to error123 302')]
#     expected_lists = [scan('302'), scan('302')]

#     for i in range(len(test_lists_good)):
#         test_list = test_lists_good[i]
#         expected_list = expected_lists[i]
#         assert_equal(parser.parse_num(test_list), *expected_list)

#     # test bad situations
#     test_lists_bad = [scan('south'), scan('door'), scan('to'), scan('error123'), scan('
east door'),
#                       scan('bear go to the door'), scan('the princess kill 10 bear'),[
]]
```

```
#     for i in range(len(test_lists_bad)):
#         test_list = test_lists_bad[i]
#         assert_equal(parser.parse_num(test_list), None)


# def test_parse_object():
#     ''' test parse_object function '''
#     parser = Parser()
#     # test good situations
#     test_lists_good =  [scan('south'), scan('door'), scan('the bear'), scan('east door'
),
#                         scan('bear go to the door'), scan('the princess kill 10 bear')
]

#     expected_lists = [scan('south'), scan('door'), scan('bear'),
#                       scan('east'), scan('bear'), scan('princess')]

#     for i in range(len(test_lists_good)):
#         test_list = test_lists_good[i]
#         expected_list = expected_lists[i]
#         assert_equal(parser.parse_object(test_list), *expected_list)

#     # test bad situations
#     test_lists_bad = [scan('go'), scan('to'), scan('234'), scan('error123'), scan('go t
o east'), []]
#     for i in range(len(test_lists_bad)):
#         test_list = test_lists_bad[i]
#         assert_raises(ParserError, parser.parse_object, test_list)


# def test_class_sentence():
#     # test good situations
#     test_lists_good =  [scan('bear go east'), scan('princess kill bear'), scan(
#                         'princess kill 10 bears')]

#     expected_nums = [1, 1, 10]
#     expected_objects = ['east', 'bear', 'bear']

#     for i in range(len(test_lists_good)):
#         test_list = test_lists_good[i]
#         test_num = expected_nums[i]
#         test_object = expected_objects[i]

#         sentence = Sentence(*test_list)
#         assert_equal(sentence.subject, test_list[0][1])
#         assert_equal(sentence.verb, test_list[1][1])
#         assert_equal(sentence.num, test_num)
#         assert_equal(sentence.object, test_object)

#     # test bad situations, for more restrict checking
#     test_lists_bad =  [scan('south'), scan('bear'), scan('go'), scan('to'),
#                        scan('the'), scan('door'), scan('bear go to the door'),
#                        scan('the princess kill 10 bears'), []]

#     for i in range(len(test_lists_good)):
#         test_list = test_lists_bad[i]
#         assert_raises(TypeError, Sentence, *test_list)


# def test_parse_subject():
#     ''' test parse_subject function '''
#     parser = Parser()

#     test_lists =  [scan('error123 eat princess'), scan('go to east'),
#                    scan('go error123 to the carbinet door'), scan('kill 10 bears')]

#     test_subjects = [scan('bear'), scan('princess'), scan('carbinet'), scan('princess')
]
#     expected_verbs = ['eat', 'go', 'go', 'kill']
#     expected_objects = ['princess', 'east', 'carbinet', 'bear']
```

```
#     expected_nums = [1, 1, 1, 10]

#     for i in range(len(test_lists)):
#         test_list = test_lists[i]
#         test_subject = test_subjects[i]
#         expected_verb = expected_verbs[i]
#         expected_object = expected_objects[i]
#         expected_num = expected_nums[i]
#         sentence = parser.parse_subject(test_list, test_subject[0])
#         assert_equal(sentence.subject, test_subject[0][1])
#         assert_equal(sentence.verb, expected_verb)
#         assert_equal(sentence.object, expected_object)
#         assert_equal(sentence.num, expected_num)


# def test_parse_sentence():
#     ''' test parse_sentence function '''
#     parser = Parser()
#     # test good situations
#     test_lists1 =  [scan('bear go to the door'),
#                     scan('the princess kill 10 bears'),
#                     scan('kill the bear')]

#     expected_subjects = ['bear', 'princess', 'player']
#     expected_verbs = ['go', 'kill', 'kill']
#     expected_objects = ['door', 'bear', 'bear']
#     expected_nums = [1, 10, 1]

#     for i in range(len(test_lists1)):
#         test_list = test_lists1[i]
#         sentence = parser.parse_sentence(test_list)
#         expected_subject = expected_subjects[i]
#         expected_verb = expected_verbs[i]
#         expected_object = expected_objects[i]
#         expected_num = expected_nums[i]
#         assert_equal(sentence.subject, expected_subject)
#         assert_equal(sentence.verb, expected_verb)
#         assert_equal(sentence.object, expected_object)
#         assert_equal(sentence.num, expected_num)

#     # test bad situations
#     test_lists2 =  [scan('234')]
#     for i in range(len(test_lists2)):
#         test_list = test_lists2[i]
#         assert_raises(ParserError, parser.parse_object, test_list)
```

HW 50

In [7]:

```
!pip install lpthw.web
```

```
Requirement already satisfied: lpthw.web in c:\python\python36\lib\site-packages

You are using pip version 9.0.1, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

In [8]:

```
!pip install flask
```

```
Requirement already satisfied: flask in c:\python\python36\lib\site-packages
Requirement already satisfied: Jinja2>=2.10.1 in c:\python\python36\lib\site-packages (fr
om flask)
Requirement already satisfied: itsdangerous>=0.24 in c:\python\python36\lib\site-packages
(from flask)
Requirement already satisfied: Werkzeug>=0.15 in c:\python\python36\lib\site-packages (fr
om flask)
Requirement already satisfied: click>=5.1 in c:\python\python36\lib\site-packages (from f
lask)
Requirement already satisfied: MarkupSafe>=0.23 in c:\python\python36\lib\site-packages (
```

```
from Jinja2>=2.10.1->flask)
```

```
You are using pip version 9.0.1, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

In [9]:

```
# Make A Project


# $ cd projects
# $ mkdir gothonweb
# $ cd gothonweb
# $ mkdir bin gothonweb tests docs templates
# $ touch gothonweb/__init__.py
# $ touch tests/__init__.py
```

In [10]:

```
# # app.py
# from flask import Flask
# from flask import render_template

# app = Flask(__name__)

# @app.route("/")
# def index():
#     greeting = "Hello World"
#     return render_template("index.html", greeting=greeting)

# if __name__ == "__main__":
#     app.run()
```

In [11]:

```
# index.html


# <html>
# <head>
# <title>Gothons Of Planet Percal #25</title>
# </head>
# <body>
# {% if greeting %}
# I just wanted to say
# <em style="color: green; font-size: 2em;">{{ greeting }}</em>.
# {% else %}
# <em>Hello</em>, world!
# {% endif %}
# </body>
# </html>
```

In [ ]:

```

```

HW 51

In [12]:

```
# # app.py


# from flask import Flask
# from flask import render_template
# from flask import request

# app = Flask(__name__)
```

```
# @app.route("/hello", methods=['POST', 'GET'])
# def index():
#     greeting = "Hello World"

#     if request.method == "POST":
#         name = request.form['name']
#         greet = request.form['greet']
#         greeting = f"{greet}, {name}"15 return render_template("index.html", greeting=g
reeting)
#     else:
#         return render_template("hello_form.html")


# if __name__ == "__main__":
#     app.run()
```

In [13]:

```
# # hello_form.html


# {% extends "layout.html" %}{% block content %}
# <h1>Fill Out This Form</h1>
# <form action="/hello" method="POST">
# A Greeting: <input type="text" name="greet">
# <br/>
# Your Name: <input type="text" name="name">
# <br/>
# <input type="submit">
# </form>
# {% endblock %}
```

In [14]:

```
# layout.html


# <html>
# <head>
# <title>Gothons From Planet Percal #25</title>
# </head>
# <body>
# {% block content %}
# {% endblock %}
# </body>
# </html>
```

In [15]:

```
# index.html

# {% extends "layout.html" %}
# {% block content %}
# {% if greeting %}
# I just wanted to say
# <em style="color: green; font-size: 2em;">{{ greeting }}</em>.
# {% else %}
# <em>Hello</em>, world!
# {% endif %}
# {% endblock %}
```

In [16]:

```
# tools.py

# from nose.tools import *
# import re

# def assert_response(resp, contains=None, matches=None, headers=None, status="200"):

#     assert status in resp.status, "Expected response %r not in %r" % (status, resp.stat
```

```
us)

#     if status == "200":
#         assert resp.data, "Response data is empty."

#     if contains:
#         assert contains in resp.data, "Response does not contain %r" % contains

#     if matches:
#         reg = re.compile(matches)
#         assert reg.matches(resp.data), "Response does not match %r" % matches

#     if headers:
#         assert_equal(resp.headers, headers)
```

In [17]:

```
# # app_test.py

# from nose.tools import *
# from app import app

# app.config['TESTING'] = True
# web = app.test_client()

# def test_index():
# rv = web.get('/', follow_redirects=True)
# assert_equal(rv.status_code, 404)

# rv = web.get('/hello', follow_redirects=True)
# assert_equal(rv.status_code, 200)
# assert_in(b"Fill Out This Form", rv.data)
# data = {'name': 'Zed', 'greet': 'Hola'}
# rv = web.post('/hello', follow_redirects=True, data=data)
# assert_in(b"Zed", rv.data)
# assert_in(b"Hola", rv.data)
```

In [ ]:

```

```