Files and Dictionaries

Specifying the starting configuration of the grid-world agent

```
def init_world():
    # make rectangle of empty cells
    for y in range(YSIZE):
        _world[y:] = [[' '] * XSIZE]

    # add some obstacles
    # obstacle 1
    _world[YSIZE-2][XSIZE-2] = 'w'
    _world[YSIZE-1][XSIZE-2] = 'w'
    _world[YSIZE-2][XSIZE-1] = 'w'
    _world[YSIZE-1][XSIZE-1] = 'w'

    # obstacle 2
    for y in [3,4]:
        for x in range(2,9):
            _world[y][x] = 'w'
    for y in [5,6,7]:
        _world[y][2] = 'w'
        _world[y][3] = 'w'
    _world[5][7] = 'w'
    _world[5][8] = 'w'
```

Wouldn't it be easier to read it from a file?

```
wwwwwwwwwwwwwwwww
w              cw
w  c            w
w           R   w
w  wwwwwww      w
w  wwwwwww      w
w  ww   ww      w
w  ww   ww  c   w
w  ww   ww      w
w               w
wc            c w
w             www
w      c      www
wwwwwwwwwwwwwwwww
```

What do we need?

- open files: give a file name and create a file object
- close files: get rid of the file object and close connection to file
- read from files: access the file content using the file object

Opening and closing files in Python

```
infile = open('somefilename','r')

# some code that reads from the infile file object

infile.close()
```

Reading from files in Python - read()

```
infile = open('somefilename','r')

# reads the whole file content into a string
wholetext = infile.read()

infile.close()
```

Reading from files in Python - readlines()

```
infile = open('somefilename','r')

# reads the whole file content into a list of
# strings corresponding to the lines
lines = infile.readlines()

infile.close()
```

Reading from files in Python - readline()

```
infile = open('somefilename','r')

# reads one line (the next available line) into a
# string
line = infile.readline()

infile.close()
```

Reading from files in Python – file objects as iterators

```
infile = open('somefilename','r')

# reads one line at a time into a string
# continues until the end of the file is reached
for line in infile :
    # do something with the line that was read
    # e.g., print it
    print line

infile.close()
```

Reading the starting configuration from a file

```
wwwwwwwwwwwwwwwww
w              cw
w  c            w
w           R   w
w  wwwwwww      w
w  wwwwwww      w
w  ww   ww      w
w  ww   ww  c   w
w  ww   ww      w
w               w
wc            c w
w             www
w      c      www
wwwwwwwwwwwwwwwww
```

Reading the starting configuration from a file

```
wwwwwwwwwwwwwwwww
w              cw
w  c            w
w           R   w
w  wwwwwww      w
w  wwwwwww      w
w  ww   ww      w
w  ww   ww  c   w
w  ww   ww      w
w               w
wc            c w
w             www
w      c      www
wwwwwwwwwwwwwwwww
```

```
def read_start_config (file) :
    world = []
    infile = open(file,'r')

    for line in infile :
        row = process_line(line)
        world.append(row)

    infile.close()
    return world
```

Reading the starting configuration from a file

```
wwwwwwwwwwwwwwwww
w              cw
w  c            w
w           R   w
w  wwwwwww      w
w  wwwwwww      w
w  ww   ww      w
w  ww   ww  c   w
w  ww   ww      w
w               w
wc            c w
w             www
w      c      www
wwwwwwwwwwwwwwwww
```

```
def process_line (line) :
    row = []

    for char in line :

        if char != '\n':
            row.append(char)

    return row
```

Exercise

- Write a function that reads in a given file and returns a list of the *different* words occurring in this file. (That is, if a word occurs two or more times in the text, it should only appear once in the list.)
- Download 'example_text.txt', which contains an excerpt of a novel by Jane Austen, and use this file to test your function.
- Hint: You may want to use the built-in string method `split`. (Check the documentation: http://docs.python.org/lib/string-methods.html)
- Hint: Remember that the operator `in` can be used to test whether a list contains a given value.

Exercise

```
def read_words (file) :
    words = []
    infile = open(file,'r')
    for line in infile :
        line = line.split()
        for word in line:
            if not (word in words):
                words.append(word)
    infile.close()
    return words
```

Exercise 2

Now, count how many times each word occurs.

Exercise 2

Now, count how many times each word occurs.

➡ need a way to associate numbers with words

Dictionaries

... are ...

- collections of objects/values (like lists)
- not ordered (unlike lists)
- accessed by key instead of position

Example: associating registered users with their password

bill : 12345

tony : FlyingCow

alan : $a$l$a$n$

nick : asel5iiagn

Dictionaries in Python

Dictionaries in Python

```
d = {'bill':'12345', 'tony':'FlyingCow', 'alan':'$a$l$a$n$'}
d = {}
d['alan']   # accessing entries
d['alan'] = 'NewPASSWD'  # changing entries
d['nick'] = 'asel5iiagn' # adding entries
del d['tony']  # deleting entries
d.has_key('bill')  # checking whether a given key exists
```

Example: reading passwords from a file

File format:

```
bill 12345
tony FlyingCow
alan $a$l$a$n$
nick asel5iiagn
```

Example: reading passwords from a file

```
d = {}
pfile = open('passwords.txt', 'r')
for line in pfile :
      name_pw_list = line.split()
      name = name_pw_list[0]
      pw = name_pw_list[1]
      d[name] = pw
pfile.close()
```

Exercise 2

Now, count how many times each word occurs.

That is:

- Write a function that reads in a given file and counts how many times each word appears in the text. Return an object that associates words (the different words in the text) with numbers (the number of times that word appears).
- Use 'example_text.txt', the excerpt from the Jane Austen novel, to test your function.

Exercise 2 – more things to do

- Write a function that prints out the dictionary in a prettier way; e.g., one word and its count per line.
- Sort the words alphabetically.
- Sort them by how often they occur.

Exercise 2 – more things to do

- Write a function that prints out the dictionary in a prettier way; e.g., one word and its count per line.
- Sort the words alphabetically.
- Sort them by how often they occur.

➔ need a way to iterate over dictionaries

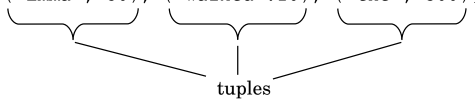
➔ need a way to sort them (by key and by value)

Making lists from dictionaries

```
d = {'Emma':50, 'the':300, 'walked':10}


d.keys()     # returns a list containing all keys


d.values()   # returns a list containing all values


d.items()    # returns a list containing all
             # key-value pairs as tuples
```

Tuples

```
>>> d = {'Emma':50, 'the':300, 'walked':10}

>>> d.items()

[('Emma', 50), ('walked':10), ('the', 300)]
```



Tuples are like lists, except that they are immutable.

Printing dictionary objects sorted by **key**

Printing dictionary objects sorted by **key**

```
d = {'Emma':50, 'the':300, 'walked':10}

k_v_pairs = d.items()
k_v_pairs.sort()
for (k,v) in k_v_pairs:
    print k, v
```

Printing dictionary objects sorted by **value**

Printing dictionary objects sorted by **value**

```
d = {'Emma':50, 'the':300, 'walked':10}

v_k_pairs = []

for (k,v) in d.items():

    v_k_pairs.append((v,k))

v_k_pairs.sort()
for (v,k) in v_k_pairs:
    print k, v
```

Exercise

- Implement a function that takes a dictionary and creates a list of the dictionary items which is sorted by the value of each key-value pair and prints out the 50 most frequent words with their frequency.
- Use example_text.txt to test your program. Read it into a dictionary using the function provided in count_words.py, then use your newly implemented function to print out the 50 most frequent words.
- Then download the texts text1.txt, text2.txt and text3.txt and look at the 50 most common words in those texts. (Don't look into the files. Just look at the 50 most common words that you get back.) What do you notice? Are there any differences? Can you make any guesses about who wrote those texts or what kind of text your are dealing with?