2. Working with Data

2.1. Lists

We’ve already seen quick introduction to lists in the previous chapter.

>>> **[**1**,** 2**,** 3**,** 4**]**

[1, 2, 3, 4]

>>> **[**"hello"**,** "world"**]**

["hello", "world"]

>>> **[**0**,** 1.5**,** "hello"**]**

[0, 1.5, "hello"]

>>> **[**0**,** 1.5**,** "hello"**]**

[0, 1.5, "hello"]

A List can contain another list as member.

>>> a = **[**1**,** 2**]**

>>> b = **[**1.5**,** 2**,** a**]**

>>> b

[1.5, 2, [1, 2]]

The built-in function range can be used to create a list of integers.

>>> range**(**4**)**

[0, 1, 2, 3]

>>> range**(**3**,** 6**)**

[3, 4, 5]

>>> range**(**2**,** 10**,** 3**)**

[2, 5, 8]

The built-in function len can be used to find the length of a list.

>>> a = **[**1**,** 2**,** 3**,** 4**]**

>>> len**(**a**)**

4

The + and \* operators work even on lists.

>>> a = **[**1**,** 2**,** 3**]**

>>> b = **[**4**,** 5**]**

>>> a + b

[1, 2, 3, 4, 5]

>>> b \* 3

[4, 5, 4, 5, 4, 5]

List can be indexed to get individual entries. Value of index can go from 0 to (length of list - 1).

>>> x = **[**1**,** 2**]**

>>> x**[**0**]**

1

>>> x**[**1**]**

2

When a wrong index is used, python gives an error.

>>> x = **[**1**,** 2**,** 3**,** 4**]**

>>> x**[**6**]**

**Traceback (most recent call last):**

File "<stdin>", line 1, in ?

IndexError: list index out of range

Negative indices can be used to index the list from right.

>>> x = **[**1**,** 2**,** 3**,** 4**]**

>>> x**[**-1**]**

4

>>> x **[**-2**]**

3

We can use list slicing to get part of a list.

>>> x = **[**1**,** 2**,** 3**,** 4**]**

>>> x**[**0**:**2**]**

[1, 2]

>>> x**[**1**:**4**]**

[2, 3, 4]

Even negative indices can be used in slicing. For example, the following examples strips the last element from the list.

>>> x**[**0**:**-1**]**

[1, 2, 3]

Slice indices have useful defaults; an omitted first index defaults to zero, an omitted second index defaults to the size of the list being sliced.

>>> x = **[**1**,** 2**,** 3**,** 4**]**

>>> a**[:**2**]**

[1, 2]

>>> a**[**2**:]**

[3, 4]

>>> a**[:]**

[1, 2, 3, 4]

An optional third index can be used to specify the increment, which defaults to 1.

>>> x = range**(**10**)**

>>> x

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> x**[**0**:**6**:**2**]**

[0, 2, 4]

We can reverse a list, just by providing -1 for increment.

>>> x**[::**-1**]**

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

List members can be modified by assignment.

>>> x = **[**1**,** 2**,** 3**,** 4**]**

>>> x**[**1**]** = 5

>>> x

[1, 5, 3, 4]

Presence of a key in a list can be tested using in operator.

>>> x = **[**1**,** 2**,** 3**,** 4**]**

>>> 2 **in** x

True

>>> 10 **in** x

False

Values can be appended to a list by calling append method on list. A method is just like a function, but it is associated with an object and can access that object when it is called. We will learn more about methods when we study classes.

>>> a = **[**1**,** 2**]**

>>> a.append**(**3**)**

>>> a

[1, 2, 3]

**Problem 1:**What will be the output of the following program?

x = **[**0**,** 1**,** **[**2**]]**

x**[**2**][**0**]** = 3

**print** x

x**[**2**]**.append**(**4**)**

**print** x

x**[**2**]** = 2

**print** x

2.1.1. The for Statement

Python provides for statement to iterate over a list. A for statement executes the specified block of code for every element in a list.

**for** x **in** **[**1**,** 2**,** 3**,** 4**]:**

**print** x

**for** i **in** range**(**10**):**

**print** i**,** i\*i**,** i\*i\*i

The built-in function zip takes two lists and returns list of pairs.

>>> zip**([**"a"**,** "b"**,** "c"**],** **[**1**,** 2**,** 3**])**

[('a', 1), ('b', 2), ('c', 3)]

It is handy when we want to iterate over two lists together.

names = **[**"a"**,** "b"**,** "c"**]**

values = **[**1**,** 2**,** 3**]**

**for** name**,** value **in** zip**(**names**,** values**):**

**print** name**,** value

**Problem 2:**Python has a built-in function sum to find sum of all elements of a list. Provide an implementation for sum.

>>> sum**([**1**,** 2**,** 3**])**

>>> 4

**Problem 3:**What happens when the above sum function is called with a list of strings? Can you make your sum function work for a list of strings as well.

>>> sum**([**"hello"**,** "world"**])**

"helloworld"

>>> sum**([**"aa"**,** "bb"**,** "cc"**])**

"aabbcc"

**Problem 4:**Implement a function product, to compute product of a list of numbers.

>>> product**([**1**,** 2**,** 3**])**

6

**Problem 5:**Write a function factorial to compute factorial of a number. Can you use the product function defined in the previous example to compute factorial?

>>> factorial**(**4**)**

24

**Problem 6:**Write a function reverse to reverse a list. Can you do this without using list slicing?

>>> reverse**([**1**,** 2**,** 3**,** 4**])**

[4, 3, 2, 1]

>>> reverse**(**reverse**([**1**,** 2**,** 3**,** 4**]))**

[1, 2, 3, 4]

**Problem 7:**Python has built-in functions min and max to compute minimum and maximum of a given list. Provide an implementation for these functions. What happens when you call your min and max functions with a list of strings?

**Problem 8:**Cumulative sum of a list [a, b, c, ...] is defined as [a, a+b, a+b+c,...]. Write a function cumulative\_sum to compute cumulative sum of a list. Does your implementation work for a list of strings?

>>> cumulative\_sum**([**1**,** 2**,** 3**,** 4**])**

[1, 3, 6, 10]

>>> cumulative\_sum**([**4**,** 3**,** 2**,** 1**])**

[4, 7, 9, 10]

**Problem 9:**Write a function cumulative\_product to compute cumulative product of a list of numbers.

>>> cumulative\_product**([**1**,** 2**,** 3**,** 4**])**

[1, 2, 6, 24]

>>> cumulative\_product**([**4**,** 3**,** 2**,** 1**])**

[4, 12, 24, 24]

**Problem 10:**Write a function *unique* to find all the unique elements of a list.

>>> unique**([**1**,** 2**,** 1**,** 3**,** 2**,** 5**])**

[1, 2, 3, 5]

**Problem 11:**Write a function *dups* to find all duplicates in the list.

>>> dups**([**1**,** 2**,** 1**,** 3**,** 2**,** 5**])**

[1, 2]

**Problem 12:**Write a function *group(list, size)* that take a list and splits into smaller lists of given size.

>>> group**([**1**,** 2**,** 3**,** 4**,** 5**,** 6**,** 7**,** 8**,** 9**],** 3**)**

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

>>> group**([**1**,** 2**,** 3**,** 4**,** 5**,** 6**,** 7**,** 8**,** 9**],** 4**)**

[[1, 2, 3, 4], [5, 6, 7, 8], [9]]

2.1.2. Sorting Lists

The sort method sorts a list in place.

>>> a = **[**2**,** 10**,** 4**,** 3**,** 7**]**

>>> a.sort**()**

>>> a

[2, 3, 4, 7 10]

The built-in function sorted returns a new sorted list without modifying the source list.

>>> a = **[**4**,** 3**,** 5**,** 9**,** 2**]**

>>> sorted**(**a**)**

[2, 3, 4, 5, 9]

>>> a

[4, 3, 5, 9, 2]

The behavior of sort method and sorted function is exactly same except that sorted returns a new list instead of modifying the given list.

The sort method works even when the list has different types of objects and even lists.

>>> a = **[**"hello"**,** 1**,** "world"**,** 45**,** 2**]**

>>> a.sort**()**

>>> a

[1, 2, 45, 'hello', 'world']

>>> a = **[[**2**,** 3**],** **[**1**,** 6**]]**

>>> a.sort**()**

>>> a

[[1, 6], [2, 3]]

We can optionally specify a function as sort key.

>>> a = **[[**2**,** 3**],** **[**4**,** 6**],** **[**6**,** 1**]]**

>>> a.sort**(**key=**lambda** x**:** x**[**1**])**

>>> a

[[6, 1], [2, 3], [4 6]]

This sorts all the elements of the list based on the value of second element of each entry.

**Problem 13:**Write a function lensort to sort a list of strings based on length.

>>> lensort**([**'python'**,** 'perl'**,** 'java'**,** 'c'**,** 'haskell'**,** 'ruby'**])**

['c', 'perl', 'java', 'ruby', 'python', 'haskell']

**Problem 14:**Improve the *unique* function written in previous problems to take an optional *key* function as argument and use the return value of the key function to check for uniqueness.

>>> unique**([**"python"**,** "java"**,** "Python"**,** "Java"**],** key=**lambda** s**:** s.lower**())**

["python", "java"]

2.2. Tuples

Tuple is a sequence type just like list, but it is immutable. A tuple consists of a number of values separated by commas.

>>> a = **(**1**,** 2**,** 3**)**

>>> a**[**0**]**

1

The enclosing braces are optional.

>>> a = 1**,** 2**,** 3

>>> a**[**0**]**

1

The built-in function len and slicing works on tuples too.

>>> len**(**a**)**

3

>>> a**[**1**:]**

2, 3

Since parenthesis are also used for grouping, tuples with a single value are represented with an additional comma.

>>> a = **(**1**)**

>> a

1

>>> b = **(**1**,)**

>>> b

(1,)

>>> b**[**0**]**

1

2.3. Sets

Sets are unordered collection of unique elements.

>>> x = set**([**3**,** 1**,** 2**,** 1**])**

set([1, 2, 3])

Python 2.7 introduced a new way of writing sets.

>>> x = **{**3**,** 1**,** 2**,** 1**}**

set([1, 2, 3])

New elements can be added to a set using the add method.

>>> x = set**([**1**,** 2**,** 3**])**

>>> x.add**(**4**)**

>>> x

set([1, 2, 3, 4])

Just like lists, the existance of an element can be checked using the in operator. However, this operation is faster in sets compared to lists.

>>> x = set**([**1**,** 2**,** 3**])**

>>> 1 **in** x

True

>>> 5 **in** x

False

**Problem 15:**Reimplement the *unique* function implemented in the earlier examples using sets.

2.4. Strings

Strings also behave like lists in many ways. Length of a string can be found using built-in function len.

>>> len**(**"abrakadabra"**)**

11

Indexing and slicing on strings behave similar to that of lists.

>>> a = "helloworld"

>>> a**[**1**]**

'e'

>>> a**[**-2**]**

'l'

>>> a**[**1**:**5**]**

"ello"

>>> a**[:**5**]**

"hello"

>>> a**[**5**:]**

"world"

>>> a**[**-2**:]**

'ld'

>>> a**[:**-2**]**

'hellowor'

>>> a**[::**-1**]**

'dlrowolleh'

The in operator can be used to check if a string is present in another string.

>>> 'hell' **in** 'hello'

True

>>> 'full' **in** 'hello'

False

>>> 'el' **in** 'hello'

True

There are many useful methods on strings.

The split method splits a string using a delimiter. If no delimiter is specified, it uses any whitespace char as delimiter.

>>> "hello world".split**()**

['hello', 'world']

>>> "a,b,c".split**(**','**)**

['a', 'b', 'c']

The join method joins a list of strings.

>>> " ".join**([**'hello'**,** 'world'**])**

'hello world'

>>> ','.join**([**'a'**,** 'b'**,** 'c'**])**

The strip method returns a copy of the given string with leading and trailing whitespace removed. Optionally a string can be passed as argument to remove characters from that string instead of whitespace.

>>> ' hello world\n'.strip**()**

'hello world'

>>> 'abcdefgh'.strip**(**'abdh'**)**

'cdefg'

Python supports formatting values into strings. Although this can include very complicated expressions, the most basic usage is to insert values into a string with the %s placeholder.

>>> a = 'hello'

>>> b = 'python'

>>> "%s %s" % **(**a**,** b**)**

'hello python'

>>> 'Chapter %d: %s' % **(**2**,** 'Data Structures'**)**

'Chapter 2: Data Structures'

**Problem 16:**Write a function extsort to sort a list of files based on extension.

>>> extsort**([**'a.c'**,** 'a.py'**,** 'b.py'**,** 'bar.txt'**,** 'foo.txt'**,** 'x.c'**])**

['a.c', 'x.c', 'a.py', 'b.py', 'bar.txt', 'foo.txt']

2.5. Working With Files

Python provides a built-in function open to open a file, which returns a file object.

f = open**(**'foo.txt'**,** 'r'**)** *# open a file in read mode*

f = open**(**'foo.txt'**,** 'w'**)** *# open a file in write mode*

f = open**(**'foo.txt'**,** 'a'**)** *# open a file in append mode*

The second argument to open is optional, which defaults to 'r' when not specified.

Unix does not distinguish binary files from text files but windows does. On windows'rb', 'wb', 'ab' should be used to open a binary file in read, write and append mode respectively.

Easiest way to read contents of a file is by using the read method.

>>> open**(**'foo.txt'**)**.read**()**

'first line\nsecond line\nlast line\n'

Contents of a file can be read line-wise using readline and readlines methods. Thereadline method returns empty string when there is nothing more to read in a file.

>>> open**(**'foo.txt'**)**.readlines**()**

['first line\n', 'second line\n', 'last line\n']

>>> f = open**(**'foo.txt'**)**

>>> f.readline**()**

'first line\n'

>>> f.readline**()**

'second line\n'

>>> f.readline**()**

'last line\n'

>>> f.readline**()**

''

The write method is used to write data to a file opened in write or append mode.

>>> f = open**(**'foo.txt'**,** 'w'**)**

>>> f.write**(**'a\nb\nc'**)**

>>> f.close**()**

>>> f.open**(**'foo.txt'**,** 'a'**)**

>>> f.write**(**'d\n'**)**

>>> f.close**()**

The writelines method is convenient to use when the data is available as a list of lines.

>>> f = open**(**'foo.txt'**)**

>>> f.writelines**([**'a\n'**,** 'b\n'**,** 'c\n'**])**

>>> f.close**()**

2.5.1. Example: Word Count

Lets try to compute the number of characters, words and lines in a file.

Number of characters in a file is same as the length of its contents.

**def** charcount**(**filename**):**

**return** len**(**open**(**filename**)**.read**())**

Number of words in a file can be found by splitting the contents of the file.

**def** wordcount**(**filename**):**

**return** len**(**open**(**filename**)**.read**()**.split**())**

Number of lines in a file can be found from readlines method.

**def** linecount**(**filename**):**

**return** len**(**open**(**filename**)**.readlines**())**

**Problem 17:**Write a program reverse.py to print lines of a file in reverse order.

$ cat she.txt

She sells seashells on the seashore;

The shells that she sells are seashells I'm sure.

So if she sells seashells on the seashore,

I'm sure that the shells are seashore shells.

$ python reverse.py she.txt

I'm sure that the shells are seashore shells.

So if she sells seashells on the seashore,

The shells that she sells are seashells I'm sure.

She sells seashells on the seashore;

**Problem 18:**Write a program to print each line of a file in reverse order.

**Problem 19:**Implement unix commands head and tail. The head and tail commands take a file as argument and prints its first and last 10 lines of the file respectively.

**Problem 20:**Implement unix command grep. The grep command takes a string and a file as arguments and prints all lines in the file which contain the specified string.

$ python grep.py she.txt sure

The shells that she sells are seashells I'm sure.

I'm sure that the shells are seashore shells.

**Problem 21:**Write a program *wrap.py* that takes filename and width as aruguments and wraps the lines longer than *width*.

$ python wrap.py she.txt 30

I'm sure that the shells are s

eashore shells.

So if she sells seashells on t

he seashore,

The shells that she sells are

seashells I'm sure.

She sells seashells on the sea

shore;

**Problem 22:**The above wrap program is not so nice because it is breaking the line at middle of any word. Can you write a new program *wordwrap.py* that works like*wrap.py*, but breaks the line only at the word boundaries?

$ python wordwrap.py she.txt 30

I'm sure that the shells are

seashore shells.

So if she sells seashells on

the seashore,

The shells that she sells are

seashells I'm sure.

She sells seashells on the

seashore;

**Problem 23:**Write a program *center\_align.py* to center align all lines in the given file.

$ python center\_align.py she.txt

I'm sure that the shells are seashore shells.

So if she sells seashells on the seashore,

The shells that she sells are seashells I'm sure.

She sells seashells on the seashore;

2.6. List Comprehensions

List Comprehensions provide a concise way of creating lists. Many times a complex task can be modelled in a single line.

Here are some simple examples for transforming a list.

>>> a = range**(**10**)**

>>> a

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> **[**x **for** x **in** a**]**

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> **[**x\*x **for** x **in** a**]**

[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> **[**x+1 **for** x **in** a**]**

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

It is also possible to filter a list using if inside a list comprehension.

>>> a = range**(**10**)**

>>> **[**x **for** x **in** a **if** x % 2 == 0**]**

[0, 2, 4, 6, 8]

>>> **[**x\*x **for** x **in** a **if** x%2 == 0**]**

[0, 4, 8, 36, 64]

It is possible to iterate over multiple lists using the built-in function zip.

>>> a = **[**1**,** 2**,** 3**,** 4**]**

>>> b = **[**2**,** 3**,** 5**,** 7**]**

>>> zip**(**a**,** b**)**

[(1, 2), (2, 3), (3, 5), (4, 7)]

>>> **[**x+y **for** x**,** y **in** zip**(**a**,** b**)]**

[3, 5, 8, 11]

we can use multiple for clauses in single list comprehension.

>>> **[(**x**,** y**)** **for** x **in** range**(**5**)** **for** y **in** range**(**5**)** **if** **(**x+y**)**%2 == 0**]**

[(0, 0), (0, 2), (0, 4), (1, 1), (1, 3), (2, 0), (2, 2), (2, 4), (3, 1), (3, 3), (4, 0), (4, 2), (4, 4)]

>>> **[(**x**,** y**)** **for** x **in** range**(**5**)** **for** y **in** range**(**5**)** **if** **(**x+y**)**%2 == 0 **and** x != y**]**

[(0, 2), (0, 4), (1, 3), (2, 0), (2, 4), (3, 1), (4, 0), (4, 2)]

>>> **[(**x**,** y**)** **for** x **in** range**(**5**)** **for** y **in** range**(**x**)** **if** **(**x+y**)**%2 == 0**]**

[(2, 0), (3, 1), (4, 0), (4, 2)]

The following example finds all Pythagorean triplets using numbers below 25. (x, y,z) is a called pythagorean triplet if x\*x + y\*y == z\*z.

>>> n = 25

>>> **[(**x**,** y**,** z**)** **for** x **in** range**(**1**,** n**)** **for** y **in** range**(**x**,** n**)** **for** z **in** range**(**y**,** n**)** **if** x\*x + y\*y == z\*z**]**

[(3, 4, 5), (5, 12, 13), (6, 8, 10), (8, 15, 17), (9, 12, 15), (12, 16, 20)]

**Problem 24:**Provide an implementation for zip function using list comprehensions.

>>> zip**([**1**,** 2**,** 3**],** **[**"a"**,** "b"**,** "c"**])**

[(1, "a"), (2, "b"), (3, "c")]

**Problem 25:**Python provides a built-in function map that applies a function to each element of a list. Provide an implementation for map using list comprehensions.

>>> **def** square**(**x**):** **return** x \* x

...

>>> map**(**square**,** range**(**5**))**

[0, 1, 4, 9, 16]

**Problem 26:**Python provides a built-in function filter(f, a) that returns items of the list a for which f(item) returns true. Provide an implementation for filter using list comprehensions.

>>> **def** even**(**x**):** **return** x %2 == 0

...

>>> filter**(**even**,** range**(**10**))**

[0, 2, 4, 6, 8]

**Problem 27:**Write a function triplets that takes a number n as argument and returns a list of triplets such that sum of first two elements of the triplet equals the third element using numbers below n. Please note that (a, b, c) and (b, a, c) represent same triplet.

>>> triplets**(**5**)**

[(1, 1, 2), (1, 2, 3), (1, 3, 4), (2, 2, 4)]

**Problem 28:**Write a function enumerate that takes a list and returns a list of tuples containing (index,item) for each item in the list.

>>> enumerate**([**"a"**,** "b"**,** "c"**])**

[(0, "a"), (1, "b"), (2, "c")]

>>> **for** index**,** value **in** enumerate**([**"a"**,** "b"**,** "c"**]):**

... **print** index**,** value

0 a

1 b

2 c

**Problem 29:**Write a function array to create an 2-dimensional array. The function should take both dimensions as arguments. Value of each element can be initialized to None:

>>> a = array**(**2**,** 3**)**

>>> a

[[None, None, None], [None, None, None]]

>>> a**[**0**][**0**]** = 5

[[5, None, None], [None, None, None]]

**Problem 30:**Write a python function parse\_csv to parse csv (comma separated values) files.

>>> **print** open**(**'a.csv'**)**.read**()**

a,b,c

1,2,3

2,3,4

3,4,5

>>> parse\_csv**(**'a.csv'**)**

[['a', 'b', 'c'], ['1', '2', '3'], ['2', '3', '4'], ['3', '4', '5']]

**Problem 31:**Generalize the above implementation of csv parser to support any delimiter and comments.

>>> **print** open**(**'a.txt'**)**.read**()**

# elements are separated by ! and comment indicator is #

a!b!c

1!2!3

2!3!4

3!4!5

>>> parse**(**'a.txt'**,** '!'**,** '#'**)**

[['a', 'b', 'c'], ['1', '2', '3'], ['2', '3', '4'], ['3', '4', '5']]

**Problem 32:**Write a function mutate to compute all words generated by a single mutation on a given word. A mutation is defined as inserting a character, deleting a character, replacing a character, or swapping 2 consecutive characters in a string. For simplicity consider only letters from a to z.

>>> words = mutate**(**'hello'**)**

>>> 'helo' **in** words

True

>>> 'cello' **in** words

True

>>> 'helol' **in** words

True

**Problem 33:**Write a function nearly\_equal to test whether two strings are nearly equal. Two strings a and b are nearly equal when a can be generated by a single mutation on b.

>>> nearly\_equal**(**'python'**,** 'perl'**)**

False

>>> nearly\_equal**(**'perl'**,** 'pearl'**)**

True

>>> nearly\_equal**(**'python'**,** 'jython'**)**

True

>>> nearly\_equal**(**'man'**,** 'woman'**)**

False

2.7. Dictionaries

Dictionaries are like lists, but they can be indexed with non integer keys also. Unlike lists, dictionaries are not ordered.

>>> a = **{**'x'**:** 1**,** 'y'**:** 2**,** 'z'**:** 3**}**

>>> a**[**'x'**]**

1

>>> a**[**'z'**]**

3

>>> b = **{}**

>>> b**[**'x'**]** = 2

>>> b**[**2**]** = 'foo'

>>> b**[(**1**,** 2**)]** = 3

>>> b

{(1, 2): 3, 'x': 2, 2: 'foo'}

The del keyword can be used to delete an item from a dictionary.

>>> a = **{**'x'**:** 1**,** 'y'**:** 2**,** 'z'**:** 3**}**

>>> **del** a**[**'x'**]**

>>> a

{'y': 2, 'z': 3}

The keys method returns all keys in a dictionary, the values method returns all values in a dictionary and items method returns all key-value pairs in a dictionary.

>>> a.keys**()**

['x', 'y', 'z']

>>> a.values**()**

[1, 2, 3]

>>> a.items**()**

[('x', 1), ('y', 2), ('z', 3)]

The for statement can be used to iterate over a dictionary.

>>> **for** key **in** a**:** **print** key

...

x

y

z

>>> **for** key**,** value **in** a.items**():** **print** key**,** value

...

x 1

y 2

z 3

Presence of a key in a dictionary can be tested using in operator or has\_key method.

>>> 'x' **in** a

True

>>> 'p' **in** a

False

>>> a.has\_key**(**'x'**)**

True

>>> a.has\_key**(**'p'**)**

False

Other useful methods on dictionaries are get and setdefault.

>>> d = **{**'x'**:** 1**,** 'y'**:** 2**,** 'z'**:** 3**}**

>>> d.get**(**'x'**,** 5**)**

1

>>> d.get**(**'p'**,** 5**)**

5

>>> d.setdefault**(**'x'**,** 0**)**

1

>>> d

{'x': 1, 'y': 2, 'z': 3}

>>> d.setdefault**(**'p'**,** 0**)**

0

>>> d

{'y': 2, 'x': 1, 'z': 3, 'p': 0}

Dictionaries can be used in string formatting to specify named parameters.

>>> 'hello %(name)s' % **{**'name'**:** 'python'**}**

'hello python'

>>> 'Chapter %(index)d: %(name)s' % **{**'index'**:** 2**,** 'name'**:** 'Data Structures'**}**

'Chapter 2: Data Structures'

2.7.1. Example: Word Frequency

Suppose we want to find number of occurrences of each word in a file. Dictionary can be used to store the number of occurrences for each word.

Lets first write a function to count frequency of words, given a list of words.

**def** word\_frequency**(**words**):**

*"""Returns frequency of each word given a list of words.*

*>>> word\_frequency(['a', 'b', 'a'])*

*{'a': 2, 'b': 1}*

*"""*

frequency = **{}**

**for** w **in** words**:**

frequency**[**w**]** = frequency.get**(**w**,** 0**)** + 1

**return** frequency

Getting words from a file is very trivial.

**def** read\_words**(**filename**):**

**return** open**(**filename**)**.read**()**.split**()**

We can combine these two functions to find frequency of all words in a file.

**def** main**(**filename**):**

frequency = word\_frequency**(**read\_words**(**filename**))**

**for** word**,** count **in** frequency.items**():**

**print** word**,** count

**if** \_\_name\_\_ == "\_\_main\_\_"**:**

**import** sys

main**(**sys.argv**[**1**])**

**Problem 34:**Improve the above program to print the words in the descending order of the number of occurrences.

**Problem 35:**Write a program to count frequency of characters in a given file. Can you use character frequency to tell whether the given file is a Python program file, C program file or a text file?

**Problem 36:**Write a program to find anagrams in a given list of words. Two words are called anagrams if one word can be formed by rearranging letters of another. For example ‘eat’, ‘ate’ and ‘tea’ are anagrams.

>>> anagrams**([**'eat'**,** 'ate'**,** 'done'**,** 'tea'**,** 'soup'**,** 'node'**])**

[['eat', 'ate', 'tea], ['done', 'node'], ['soup']]

**Problem 37:**Write a function valuesort to sort values of a dictionary based on the key.

>>> valuesort**({**'x'**:** 1**,** 'y'**:** 2**,** 'a'**:** 3**})**

[3, 1, 2]

**Problem 38:**Write a function invertdict to interchange keys and values in a dictionary. For simplicity, assume that all values are unique.

>>> invertdict**({**'x'**:** 1**,** 'y'**:** 2**,** 'z'**:** 3**})**

{1: 'x', 2: 'y', 3: 'z'}

2.7.2. Understanding Python Execution Environment

Python stores the variables we use as a dictionary. The globals() function returns all the globals variables in the current environment.

>>> globals**()**

{'\_\_builtins\_\_': <module '\_\_builtin\_\_' (built-in)>, '\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None}

>>> x = 1

>>> globals**()**

{'\_\_builtins\_\_': <module '\_\_builtin\_\_' (built-in)>, '\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, 'x': 1}

>>> x = 2

>>> globals**()**

{'\_\_builtins\_\_': <module '\_\_builtin\_\_' (built-in)>, '\_\_name\_\_': '\_\_main\_\_', '\_\_doc\_\_': None, 'x': 2}

>>> globals**()[**'x'**]** = 3

>>> x

3

Just like globals python also provides a function locals which gives all the local variables in a function.

>>> **def** f**(**a**,** b**):** **print** locals**()**

...

>>> f**(**1**,** 2**)**

{'a': 1, 'b': 2}

One more example:

>>> **def** f**(**name**):**

... **return** "Hello %(name)s!" % locals**()**

...

>>> f**(**"Guido"**)**

Hello Guido!

**Further Reading:**

* The article [A Plan for Spam](http://paulgraham.com/spam.html) by [Paul Graham](http://paulgraham.com/) describes a method of detecting spam using probability of occurrence of a word in spam.

About this Book

This book is prepared from the training notes of [Anand Chitipothu](http://anandology.com/).

Anand conducts Python trainings classes on a semi-regular basis in Bangalore, India. Checkout out the[upcoming trainings](http://anandology.com/trainings/) if you are interested.

[Table Of Contents](http://anandology.com/python-practice-book/index.html)

* [2. Working with Data](http://anandology.com/python-practice-book/working-with-data.html)
  + [2.1. Lists](http://anandology.com/python-practice-book/working-with-data.html#lists)
    - [2.1.1. The for Statement](http://anandology.com/python-practice-book/working-with-data.html#the-for-statement)
    - [2.1.2. Sorting Lists](http://anandology.com/python-practice-book/working-with-data.html#sorting-lists)
  + [2.2. Tuples](http://anandology.com/python-practice-book/working-with-data.html#tuples)
  + [2.3. Sets](http://anandology.com/python-practice-book/working-with-data.html#sets)
  + [2.4. Strings](http://anandology.com/python-practice-book/working-with-data.html#strings)
  + [2.5. Working With Files](http://anandology.com/python-practice-book/working-with-data.html#working-with-files)
    - [2.5.1. Example: Word Count](http://anandology.com/python-practice-book/working-with-data.html#example-word-count)
  + [2.6. List Comprehensions](http://anandology.com/python-practice-book/working-with-data.html#list-comprehensions)
  + [2.7. Dictionaries](http://anandology.com/python-practice-book/working-with-data.html#dictionaries)
    - [2.7.1. Example: Word Frequency](http://anandology.com/python-practice-book/working-with-data.html#example-word-frequency)
    - [2.7.2. Understanding Python Execution Environment](http://anandology.com/python-practice-book/working-with-data.html#understanding-python-execution-environment)

Related Topics

* [Documentation overview](http://anandology.com/python-practice-book/index.html)
  + Previous: [1. Getting Started](http://anandology.com/python-practice-book/getting-started.html)
  + Next: [3. Modules](http://anandology.com/python-practice-book/modules.html)

This Page

* [Show Source](http://anandology.com/python-practice-book/_sources/working-with-data.txt)

Quick search

Top of Form



Bottom of Form

Enter search terms or a module, class or function name.

© Copyright 2012, [Anand Chitipothu](http://anandology.com/). Created using [Sphinx](http://sphinx.pocoo.org/).