wtech
Teknolojide Kadın Derneği

Introduction to Machine Learning with Python

Dictionaries, file operations and custom functions

Dr. Süha Tuna
İTÜ Informatics Institute

Dictionaries

A dictionary is a more general version of a list. Here is a list that contains the number of days in the months of the year:

```
days = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
```

If we want the number of days in January, use `days[0]`. December is `days[11]` or `days[-1]`.

Here is a dictionary of the days in the months of the year:

```
days = {'January':31, 'February':28, 'March':31, 'April':30,
        'May':31, 'June':30, 'July':31, 'August':31,
        'September':30, 'October':31, 'November':30, 'December':31}
```

To get the number of days in January, we use `days['January']`. One benefit of using dictionaries here is the code is more readable, and we don't have to figure out which index in the list a given month is at. Dictionaries have a number of other uses, as well.

Basics

Changing dictionaries Let's start with this dictionary:

```
d = {'A':100, 'B':200}
```

- To change `d['A']` to 400, do

  ```
  d['A']=400
  ```

- To add a new entry to the dictionary, we can just assign it, like below:

  ```
  d['C']=500
  ```

  Note that this sort of thing does not work with lists. Doing `L[2]=500` on a list with two elements would produce an index out of range error. But it does work with dictionaries.

- To delete an entry from a dictionary, use the `del` operator:

  ```
  del d['A']
  ```

Empty dictionary The empty dictionary is `{}`, which is the dictionary equivalent of `[]` for lists or `''` for strings.

Important note The order of items in a dictionary will not necessarily be the order in which put them into the dictionary. Internally, Python rearranges things in a dictionary in order to optimize performance.

Dictionary examples

Example 1 You can use a dictionary as an actual dictionary of definitions:

```
d = {'dog' : 'has a tail and goes woof!',
     'cat' : 'says meow',
     'mouse' : 'chased by cats'}
```

Here is an example of the dictionary in use:

```
word = input('Enter a word: ')
print('The definition is:', d[word])
```

```
Enter a word: mouse
The definition is: chased by cats
```

Example 2 The following dictionary is useful in a program that works with Roman numerals.

```
numerals = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}
```

Example 3 In the game Scrabble, each letter has a point value associated with it. We can use the following dictionary for the letter values:

```
points = {'A':1, 'B':3, 'C':3, 'D':2, 'E':1, 'F':4, 'G':2,
          'H':4, 'I':1, 'J':8, 'K':5, 'L':1, 'M':3, 'N':1,
          'O':1, 'P':3, 'Q':10, 'R':1, 'S':1, 'T':1, 'U':1,
          'V':4, 'W':4, 'X':8, 'Y':4, 'Z':10}
```

Dictionary examples

To score a word, we can do the following:

```
score = sum([points[c] for c in word])
```

Or, if you prefer the long way:

```
total = 0
for c in word:
    total += points[c]
```

Example 4 A dictionary provides a nice way to represent a deck of cards:

```
deck = [{'value':i, 'suit':c}
        for c in ['spades', 'clubs', 'hearts', 'diamonds']
        for i in range(2,15)]
```

The deck is actually a list of 52 dictionaries. The `shuffle` method can be used to shuffle the deck:

```
shuffle(deck)
```

The first card in the deck is `deck[0]`. To get the value and the suit of the card, we would use the following:

```
deck[0]['value']
deck[0]['suit']
```

Working with dictionaries

Copying dictionaries Just like for lists, making copies of dictionaries is a little tricky for reasons we will cover later. To copy a dictionary, use its `copy` method. Here is an example:

```
d2 = d.copy()
```

`in` The `in` operator is used to tell if something is a key in the dictionary. For instance, say we have the following dictionary:

```
d = {'A':100, 'B':200}
```

Referring to a key that is not in the dictionary will produce an error. For instance, `print(d['C'])` will fail. To prevent this error, we can use the `in` operator to check first if a key is in the dictionary before trying to use the key. Here is an example:

```
letter = input('Enter a letter: ')
if letter in d:
    print('The value is', d[letter])
else:
    print('Not in dictionary')
```

You can also use `not in` to see if a key is not in the dictionary.

Working with dictionaries

Looping Looping through dictionaries is similar to looping through lists. Here is an example that prints the keys in a dictionary:

```
for key in d:
    print(key)
```

Here is an example that prints the values:

```
for key in d:
    print(d[key])
```

Lists of keys and values The following table illustrates the ways to get lists of keys and values from a dictionary. It uses the dictionary `d={'A':1,'B':3}`.

Statement	Result	Description
`list(d)`	`['A','B']`	keys of `d`
`list(d.values())`	`[1,3]`	values of `d`
`list(d.items())`	`[('A',1),('B',3)]`	(key,value) pairs of `d`

The pairs returned by `d.items` are called *tuples*. Tuples are a lot like lists. They are covered in Section 19.2.

Here is a use of `d.items` to find all the keys in a dictionary `d` that correspond to a value of 100:

```
d = {'A':100, 'B':200, 'C':100}
L = [x[0] for x in d.items() if x[1]==100]
```

```
['A', 'C']
```

Working with dictionaries: `dict`

`dict` The `dict` function is another way to create a dictionary. One use for it is kind of like the opposite of the `items` method:

```
d = dict([('A',100),('B',300)])
```

This creates the dictionary `{'A':100,'B':300}`. This way of building a dictionary is useful if your program needs to construct a dictionary while it is running.

Dictionary comprehensions Dictionary comprehensions work similarly to list comprehensions. The following simple example creates a dictionary from a list of words, where the values are the lengths of the words:

```
d = {s : len(s) for s in words}
```

Exercises

1. Write a program that repeatedly asks the user to enter product names and prices. Store all of these in a dictionary whose keys are the product names and whose values are the prices. When the user is done entering products and prices, allow them to repeatedly enter a product name and print the corresponding price or a message if the product is not in the dictionary.

2. Using the dictionary created in the previous problem, allow the user to enter a dollar amount and print out all the products whose price is less than that amount.

3. For this problem, use the dictionary from the beginning of this chapter whose keys are month names and whose values are the number of days in the corresponding months.

   (a) Ask the user to enter a month name and use the dictionary to tell them how many days are in the month.

   (b) Print out all of the keys in alphabetical order.

   (c) Print out all of the months with 31 days.

   (d) Print out the (key-value) pairs sorted by the number of days in each month

Exercises

4. Write a program that uses a dictionary that contains ten user names and passwords. The program should ask the user to enter their username and password. If the username is not in the dictionary, the program should indicate that the person is not a valid user of the system. If the username is in the dictionary, but the user does not enter the right password, the program should say that the password is invalid. If the password is correct, then the program should tell the user that they are now logged in to the system.

5. Repeatedly ask the user to enter a team name and the how many games the team won and how many they lost. Store this information in a dictionary where the keys are the team names and the values are lists of the form `[`*wins*`,` *losses*`]`.

   (a) Using the dictionary created above, allow the user to enter a team name and print out the team's winning percentage.

   (b) Using the dictionary, create a list whose entries are the number of wins of each team.

   (c) Using the dictionary, create a list of all those teams that have winning records.

6. Repeatedly ask the user to enter game scores in a format like *team1 score1 - team2 score2*. Store this information in a dictionary where the keys are the team names and the values are lists of the form `[`*wins*`,` *losses*`]`.

7. Create a 5×5 list of numbers. Then write a program that creates a dictionary whose keys are the numbers and whose values are the how many times the number occurs. Then print the three most common numbers.

8. Using the card dictionary from earlier in this chapter, create a simple card game that deals two players three cards each. The player with the highest card wins. If there is a tie, then compare the second highest card and, if necessary, the third highest. If all three cards have the same value, then the game is a draw.

Reading from files

Suppose we have a text file called `example.txt` whose contents are shown below, and we want to read its contents into Python. There are several ways to do so. We will look at two of them.

```
Hello.
This is a text file.
Bye!
```

1. The first way to read a text file uses a list comprehension to load the file line-by-line into a list:

   ```
   lines = [line.strip() for line in open('example.txt')]
   ```

   The list `lines` is now

   ```
   ['Hello.', 'This is a text file.', 'Bye!']
   ```

   The string method `strip` removes any whitespace characters from the beginning and end of a string. If we had not used it, each line would contain a newline character at the end of the line. This is usually not what we want.

   Note: `strip` removes whitespace from both the beginning and end of the line. Use `rstrip` if you need to preserve whitespace at the beginning of the line.

2. The second way of reading a text file loads the entire file into a string:

   ```
   s = open('example.txt').read()
   ```

   The string `s` is now

   ```
   'Hello.\nThis is a text file.\nBye!'
   ```

Reading from files

Say your program opens a file, like below:

```
s = open('file.txt').read()
```

The file is assumed to be in the same directory as your program itself. If it is in a different directory, then you need to specify that, like below:

```
s = open('c:/users/heinold/desktop/file.txt').read()
```

Writing to files

```
f = open('writefile.txt', 'w')
print('This is line 1.', file=f)
print('This is line 2.', file=f)
f.close()
```

Examples

Example 1 Write a program that reads a list of temperatures from a file called `temps.txt`, converts those temperatures to Fahrenheit, and writes the results to a file called `ftemps.txt`.

```
file1 = open('ftemps.txt', 'w')
temperatures = [line.strip() for line in open('temps.txt')]
for t in temperatures:
    print(int(t)*9/5+32, file=file1)
file1.close()
```

Example 2 In Section 7.6 we wrote a simple quiz game. The questions and answers were both contained in lists hard-coded into the program. Instead of that, we can store the questions and answers in files. That way, if you decide to change the questions or answers, you just have to change their files. Moreover, if you decide to give the program to someone else who doesn't know

Python, they can easily create their own lists of questions and answers. To do this, we just replace the lines that create the lists with the following:

```
questions = [line.strip() for line in open('questions.txt')]
answers = [line.strip() for line in open('answers.txt')]
```

A file operations example: Wordplay

Assuming the wordlist file is `wordlist.txt`, we can load the words into a list using the line below.

```
wordlist = [line.strip() for line in open('wordlist.txt')]
```

Example 1 Print all three letter words.

```
for word in wordlist:
    if len(word)==3:
        print(word)
```

Note that this and most of the upcoming examples can be done with list comprehensions:

```
print([word for word in wordlist if len(word)==3])
```

Example 2 Print all the words that start with `gn` or `kn`.

```
for word in wordlist:
    if word[:2]=='gn' or word[:2]=='kn':
        print(word)
```

Example 3 Determine what percentage of words start with a vowel.

```
count = 0
for word in wordlist:
    if word[0] in 'aeiou':
        count=count+1
print(100*count/len(wordlist))
```

A file operations example: Wordplay

Example 4 Print all 7-letter words that start with `th` and end in `ly`. Things like this are good for cheating at crosswords.

```
for word in wordlist:
    if len(word)==7 and word[:2]=='th' and word[-2:]=='ly':
        print(word)
```

Example 5 Print the first ten words that start with `q`.

```
i=0
while wordlist[i][0]!='q':
    i=i+1
print(wordlist[i:i+10]
```

Exercises

1. You are given a file called `class_scores.txt`, where each line of the file contains a one-word username and a test score separated by spaces, like below:.

```
GWashington 83
JAdams 86
```

   Write code that scans through the file, adds 5 points to each test score, and outputs the usernames and new test scores to a new file, `scores2.txt`.

2. You are given a file called `grades.txt`, where each line of the file contains a one-word student username and three test scores separated by spaces, like below:.

```
GWashington 83 77 54
JAdams 86 69 90
```

   Write code that scans through the file and determines how many students passed all three tests.

3. You are given a file called `logfile.txt` that lists log-on and log-off times for users of a system. A typical line of the file looks like this:

```
Van Rossum, 14:22, 14:37
```

Each line has three entries separated by commas: a username, a log-on time, and a log-off time. Times are given in 24-hour format. You may assume that all log-ons and log-offs occur within a single workday.

Write a program that scans through the file and prints out all users who were online for at least an hour.

Exercises

4. You are given a file called `students.txt`. A typical line in the file looks like:

```
walter melon        melon@email.msmary.edu        555-3141
```

   There is a name, an email address, and a phone number, each separated by tabs. Write a program that reads through the file line-by-line, and for each line, capitalizes the first letter of the first and last name and adds the area code 301 to the phone number. Your program should write this to a new file called `students2.txt`. Here is what the first line of the new file should look like:

```
Walter Melon        melon@email.msmary.edu        301-555-3141
```

5. You are given a file `namelist.txt` that contains a bunch of names. Some of the names are a first name and a last name separated by spaces, like *George Washington*, while others have a middle name, like *John Quincy Adams*. There are no names consisting of just one word or more than three words. Write a program that asks the user to enter initials, like *GW* or *JQA*, and prints all the names that match those initials. Note that initials like *JA* should match both *John Adams* and *John Quincy Adams*.

6. You are given a file `namelist.txt` that contains a bunch of names. Print out all the names in the list in which the vowels *a*, *e*, *i*, *o*, and *u* appear in order (with repeats possible). The first vowel in the name must be *a* and after the first *u*, it is okay for there to be other vowels. An example is *Ace Elvin Coulson*.

7. You are given a file called `baseball.txt`. A typical line of the file starts like below.

```
Ichiro Suzuki     SEA     162     680     74    ...[more stats]
```

   Each entry is separated by a tab, `\t`. The first entry is the player's name and the second is their team. Following that are 16 statistics. Home runs are the seventh stat and stolen bases are the eleventh. Print out all the players who have at least 20 home runs and at least 20 stolen bases.

Functions

```
def print_hello():
    print('Hello!')

print_hello()
print('1234567')
print_hello()
```

```
Hello!
1234567
Hello!
```

```
def draw_square():
    print('*' * 15)
    print('*', ' '*11, '*')
    print('*', ' '*11, '*')
    print('*' * 15)
```

Arguments

```
def print_hello(n):
    print('Hello ' * n)
    print()

print_hello(3)
print_hello(5)
times = 2
print_hello(times)
```

```
Hello Hello Hello
Hello Hello Hello Hello Hello
Hello Hello
```

```
def multiple_print(string, n):
    print(string * n)
    print()

multiple_print('Hello', 5)
multiple_print('A', 10)
```

```
HelloHelloHelloHelloHello
AAAAAAAAAA
```

Returning values

Example 1 Here is a simple function that converts temperatures from Celsius to Fahrenheit.

```
def convert(t):
    return t*9/5+32

print(convert(20))
```

```
68
```

The `return` statement is used to send the result of a function's calculations back to the caller.

Notice that the function itself does not do any printing. The printing is done outside of the function. That way, we can do math with the result, like below.

```
print(convert(20)+5)
```

Example 2 As another example, the Python `math` module contains trig functions, but they only work in radians. Let us write our own sine function that works in degrees.

```
from math import pi, sin

def deg_sin(x):
    return sin(pi*x/180)
```

Returning values

Example 4 A `return` statement by itself can be used to end a function early.

```
def multiple_print(string, n, bad_words):
    if string in bad_words:
        return
    print(string * n)
    print()
```

The same effect can be achieved with an if/else statement, but in some cases, using `return` can make your code simpler and more readable.

Default arguments and keyword arguments

```
def multiple_print(string, n=1)
    print(string * n)
    print()

multiple_print('Hello', 5)
multiple_print('Hello')
```

```
HelloHelloHelloHelloHello
Hello
```

Default arguments need to come at the end of the function definition, after all of the non-default arguments.

Keyword arguments A related concept to default arguments is *keyword arguments*. Say we have the following function definition:

```
def fancy_print(text, color, background, style, justify):
```

Every time you call this function, you have to remember the correct order of the arguments. Fortunately, Python allows you to name the arguments when calling the function, as shown below:

```
fancy_print(text='Hi', color='yellow', background='black',
            style='bold', justify='left')
```

```
fancy_print(text='Hi', style='bold', justify='left',
            background='black', color='yellow')
```

Default arguments and keyword arguments

```
def fancy_print(text, color='black', background='white',
                style='normal', justify='left'):
    # function code goes here

fancy_print('Hi', style='bold')
fancy_print('Hi', color='yellow', background='black')
fancy_print('Hi')
```

Local variables

```
def func1():
    for i in range(10):
        print(i)

def func2():
    i=100
    func1()
    print(i)
```

```
def reset():
    global time_left
    time_left = 0

def print_time():
    print(time_left)

time_left=30
```

Local variables

Arguments We finish the chapter with a bit of a technical detail. You can skip this section for the time being if you don't want to worry about details right now. Here are two simple functions:

```
def func1(x):
    x = x + 1

def func2(L):
    L = L + [1]

a=3
M=[1,2,3]
func1(a)
func2(M)
```

When we call `func1` with `a` and `func2` with `L`, a question arises: do the functions change the values of `a` and `L`? The answer may surprise you. The value of `a` is unchanged, but the value of `L` is changed. The reason has to do with a difference in the way that Python handles numbers and lists. Lists are said to be *mutable* objects, meaning they can be changed, whereas numbers and strings are *immutable*, meaning they cannot be changed. There is more on this in Section 19.1.

Exercises

1. Write a function called `rectangle` that takes two integers `m` and `n` as arguments and prints out an $m \times n$ box consisting of asterisks. Shown below is the output of `rectangle(2, 4)`

   ```
   ****
   ****
   ```

2. (a) Write a function called `add_excitement` that takes a list of strings and adds an exclamation point (`!`) to the end of each string in the list. The program should modify the original list and not return anything.

   (b) Write the same function except that it should not modify the original list and should instead return a new list.

3. Write a function called `sum_digits` that is given an integer `num` and returns the sum of the digits of `num`.

4. The *digital root* of a number n is obtained as follows: Add up the digits n to get a new number. Add up the digits of that to get another new number. Keep doing this until you get a number that has only one digit. That number is the digital root.

   For example, if $n = 45893$, we add up the digits to get $4 + 5 + 8 + 9 + 3 = 29$. We then add up the digits of 29 to get $2 + 9 = 11$. We then add up the digits of 11 to get $1 + 1 = 2$. Since 2 has only one digit, 2 is our digital root.

   Write a function that returns the digital root of an integer n. [Note: there is a shortcut, where the digital root is equal to n mod 9, but do not use that here.]

5. Write a function called `first_diff` that is given two strings and returns the first location in which the strings differ. If the strings are identical, it should return -1.

6. Write a function called `binom` that takes two integers n and k and returns the binomial coefficient $\binom{n}{k}$. The definition is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.

Exercises

7. Write a function that takes an integer *n* and returns a random integer with exactly *n* digits. For instance, if *n* is 3, then 125 and 593 would be valid return values, but 093 would not because that is really 93, which is a two-digit number.

8. Write a function called `number_of_factors` that takes an integer and returns how many factors the number has.

9. Write a function called `factors` that takes an integer and returns a list of its factors.

10. Write a function called `closest` that takes a list of numbers `L` and a number `n` and returns the largest element in `L` that is not larger than `n`. For instance, if `L=[1, 6, 3, 9, 11]` and `n=8`, then the function should return 6, because 6 is the closest thing in `L` to 8 that is not larger than 8. Don't worry about if all of the things in `L` are smaller than `n`.

11. Write a function called `matches` that takes two strings as arguments and returns how many matches there are between the strings. A match is where the two strings have the same character at the same index. For instance, `'python'` and `'path'` match in the first, third, and fourth characters, so the function should return 3.

12. Recall that if `s` is a string, then `s.find('a')` will find the location of the *first a* in `s`. The problem is that it does not find the location of every *a*. Write a function called `findall` that given a string and a single character, returns a list containing all of the locations of that character in the string. It should return an empty list if there are no occurrences of the character in the string.

13. Write a function called `change_case` that given a string, returns a string with each upper case letter replaced by a lower case letter and vice-versa.

14. Write a function called `is_sorted` that is given a list and returns `True` if the list is sorted and `False` otherwise.