Outline

1. Files and Databases
   - mass storage
   - hash functions
2. Dictionaries
   - logical key values
   - nested tables
3. Persistent Data
   - storing information between executions
   - using DBM files
4. Rule Based Programming
   - storing rules in dictionaries
5. Summary + Assignments

MCS 260 Lecture 7
Introduction to Computer Science
Jan Verschelde, 27 January 2016

mass storage
dictionaries in Python

Mass Storage

tapes and disks

Mass storage means

1. the data is persistent
2. large capacity: giga or terabytes

We distinguish modes of access:

- *sequential* access: one must rewind tapes
- *direct* access: read disks from any position

We distinguish two different technologies:

- a *magnetic* file covers disk and tape surfaces
- optical disc media rely on *laser* technology

Compression software also helps increasing capacity.

Units to measure Capacity

1 byte = 8 bits. Large quantities are expressed in thousands (kilo), millions (mega), billions (giga), and trillions (tera).

units	value	value in full
Kb = kilobyte	$2^{10} \approx 10^3$	$1,024$
Mb = megabyte	$2^{20} \approx 10^6$	$1,048,576$
Gb = gigabyte	$2^{30} \approx 10^9$	$1,073,741,824$
Tb = terabyte	$2^{40} \approx 10^{12}$	$1,099,511,627,776$

The same prefixes (kilo, mega, giga, tera) measure clock speed of the CPU, or other frequencies.

$$
\begin{array}{rcl}
1 \text{ hertz} & = & 1 \text{ cycle per second} \\
1 \text{ kilohertz} & = & 2^{10} \text{ cycles per second} \\
1 \text{ megahertz} & = & 2^{20} \text{ cycles per second} \\
1 \text{ gigahertz} & = & 2^{30} \text{ cycles per second}
\end{array}
$$

Disk Organization

platters, tracks, sectors, cylinders

- A disk consists of a number of horizontal *platters*, covered by a magnetic coating.
  Data is stored on the two surfaces of each platter.
- *Tracks* are concentric circles on a surface.
  Sectors are track segments of equal size.
  Disk formatting: writing start and end of sectors.
- A *cylinder* is a set of tracks equidistant from the center of all surfaces. Consecutive data is placed in sequence on the same cylinder.
- Disks rotate and there is one moving read/write head per surface.
  An *input/output block* is a group of contiguous data read or written in one single input/output operation.

I/O Disk Operations

reading from and writing information to disk

- A *buffer* in main memory holds the entire block of data prior to writing to or after being read from disk.
- The *seek* is the movement of the heads towards the required track. The *seek time* is the time of a seek.
- The *latency time* is the time to wait for the required sector to pass beneath the read/write head. On average this equals half the *rotation time*.
- Time needed for one i/o operation:

$$t_{\mathrm{i/o}} = t_{\mathrm{seek}} + t_{\mathrm{latency}} + t_{\mathrm{transfer}}.$$

Flash Drives

the memory stick

Commonly used portable mass storage.

- connect to USB port, which powers the drive
  USB = Universal Serial Bus
- capacity goes to several gigabytes
- sends electronic signals to chambers of silicon dioxide, altering the characteristics of small electronic circuits

Advantages and disadvantages:

+ unlike a disk drive, there is no movement, sometimes faster than optical disks
– can sustain only limited number of write and erase cycles

mass storage
dictionaries in Python

1. Files and Databases
   - mass storage
   - hash functions
2. Dictionaries
   - logical key values
   - nested tables
3. Persistent Data
   - storing information between executions
   - using DBM files
4. Rule Based Programming
   - storing rules in dictionaries
5. Summary + Assignments

File Organization
records and blocks

- Data is organized in *logical records*.

  One record in a phone book has three fields:

name	address	phone number

- An input/output block can contain several records.
- The usage factor is

$$\frac{\text{\# bytes allocated to logical records}}{\text{\# bytes of physical blocks on file}}$$

Sequential File Organization

order records sequentially

- Every record on file has a *key*.
  Records are stored in order of the keys.

  In a phone book, with names sorted alphabetically,
  the key is usually the name.

- Binary search is an efficient way to search through a sorted data collection.

- The main problem with sequential file organization is the insertion of new elements.

- Solutions to this problems are
  1. store changes in a separate file that is then periodically merged with the main file
  2. leave free blocks between records
  3. use an overflow zone to insert new data

Hash-based File Organization

order of records is computed

- Keys are generated by a *hash algorithm*.

  The hash algorithm defines a *hash function*,
  mapping logical key values (like a name) to a physical address (or a position).

  Goal: even distribution of keys over addresses.

- Mapping names into addresses via combinations of the ASCII codes of the characters in the strings representing the names is a first step.

- Advantage: fast access, reduced search speed.
  Disadvantage: two different key values could be mapped to the same address.

mass storage
dictionaries in Python

Using Dictionaries

choosing a key as index

- To select from a list or tuples, the index must be a number. But very often, we list data using names as indices. Consider for example a telephone directory.

- A dictionary is an unordered set of *key:value* pairs, where *value* can be of any data type. The type of `key` must admit an ordering, it must be “*hashable*”.

For example, list summer sales according to month:

```
>>> sales = { 'jun':123, 'aug':342, 'sep' : 212 }
>>> sales
{'jun': 123, 'aug': 342, 'sep': 212}
>>> sales['aug']
342
```

Operations on Dictionaries

Modifying a dictionary:

```
>>> sales
{'jun': 123, 'aug': 342, 'sep': 212}
>>> sales['jun'] = 321
>>> sales
{'jun': 321, 'aug': 342, 'sep': 212}
```

Order a dictionary:

```
>>> sales
{'jun': 321, 'aug': 342, 'sep': 212}
>>> sales.keys()
['jun', 'aug', 'sep']
>>> ind = sales.keys()
>>> sales[ind[2]]
212
```

By assigning the keys to an ordered list `ind`, we have placed an order on the dictionary.

Deleting and Adding

Example continued ...

```
>>> sales.values()
[321, 342, 212]
>>> sales.keys()
['jun', 'aug', 'sep']
>>> len(sales)
3
```

on holiday in August ... delete August sales

```
>>> del sales['aug']
>>> sales
{'jun': 321, 'sep': 212}
>>> len(sales)
2
```

We continued in October ... add October sales:

```
>>> sales['oct'] = 99
>>> sales
{'jun': 321, 'oct': 99, 'sep': 212}
```

mass storage
dictionaries in Python

Mileage Tables – an application of dictionaries

- A mileage table lists the number of miles between several major cities.
- We store the distance between Chicago and 3 cities: Los Angeles, Miami, and New York, in a dictionary.

  The result of `input()` can immediately be used as the key to query the dictionary.
- running the program mileage.py:

```
$ python mileage.py
Give a city : Miami
Chicago - Miami : 1237 miles
$
```

Mileage Tables – distances between cities

The program saved as mileage.py:

```
# L-7 MCS 260 : a mileage table
"""
A dictionary records the distance from Chicago
to 3 other cities.  The result of a raw input
is key to query the distances in the dictionary.
"""
CHICAGO = {'Los Angeles' : 2047, 'Miami' : 1237, \
    'New York' : 807}
CITY = input('Give a city : ')
print('Chicago -', CITY, ':', CHICAGO[CITY], 'miles')
```

Nested Dictionaries – more useful mileage tables

- Mileage tables are two dimensional: we use two cities as index and obtain on return the distance.
- Build a dictionary `DISTANCE` and query it as `DISTANCE[CITY1][CITY2]` where `CITY1` and `CITY2` are 2 strings, holding the names of 2 cities.
- running the program miletab.py:

```
$ python miletab.py
Give first city : Los Angeles
Give second city : Miami
Los Angeles - Miami : 2780 miles
$
```

Nesting Dictionaries – a 4-by-4 mileage table

The name DISTANCE refers to a dictionary of dictionaries:

```
DISTANCE = { \
  'Chicago' : {'Los Angeles' : 2047, \
      'Miami' : 1237, 'New York' : 807},  \
  'Los Angeles' : {'Chicago' : 2047, \
      'Miami' : 2780, 'New York' : 2787}, \
  'Miami' : {'Chicago' : 1237, \
      'Los Angeles' : 2780, 'New York' : 1346}, \
  'New York' : {'Chicago' : 807, \
      'Los Angeles' : 2787, 'Miami' : 1346} \
}
```

```
>>> distance['Los Angeles']['Miami']
2780
```

miletab.py – the complete code

```
DISTANCE = { \
  'Chicago' : {'Los Angeles' : 2047, \
      'Miami' : 1237, 'New York' : 807},  \
  'Los Angeles' : {'Chicago' : 2047, \
      'Miami' : 2780, 'New York' : 2787}, \
  'Miami' : {'Chicago' : 1237, \
      'Los Angeles' : 2780, 'New York' : 1346}, \
  'New York' : {'Chicago' : 807, \
      'Los Angeles' : 2787, 'Miami' : 1346} \
}
CITY1 = input('Give first city : ')
CITY2 = input('Give second city : ')
print CITY1 , '-', CITY2 , ':' , \
    DISTANCE[CITY1][CITY2] , 'miles'
```

mass storage
dictionaries in Python

Persistent Data

storing information between executions

Data that is *persistent* outlives programs.

Objects constructed by a script are lost
as soon as the script ends.

Two extremes to make data persistent:

1. files: store string representations,
2. MySQL: store data in tables in a database.

Intermediate solution: DBM files.

mass storage
dictionaries in Python

using DBM files

DBM files are standard in the Python library (Python2: `anydbm`).

```
$ python
>>> import dbm
>>> libdb = dbm.open('library','c')
```

opened a new dbm with read-write access (flag = `'c'`).

Some issues about the location of files...

- The file `library` will be in the current working directory.
- If you have no permissions to write in the current directory, then opening a dbm with read-write access will fail. (Use `os.getcwd()` to see the current working directory.)
- To create the dbm file `lib` in the `/tmp` directory:

```
>>> libdb = dbm.open('/tmp/lib','c')
```

adding data to a DBM file

```
>>> libdb['0'] = str({'author':'Miller & Ranum',
... 'title':'Python Programming'})
```

keys and values must be of type string

```
>>> libdb.keys()
['0']
>>> libdb.values()
["{'title': 'Python Programming', 'author': 'Miller &
$ ls
```

$\rightarrow$ `library` is a file in current directory.

adding and selecting books using the key

```
>>> import dbm
>>> mylib = dbm.open('library','c')
>>> mylib.keys()
['0']
>>> mylib['1'] = str({'author':'Brookshear',
... 'title':'Computer Science: an overview'})
>>> mylib.values()
["{'title': 'Python Programming', 'author': 'Miller & Ranum
 "{'title': 'Computer Science: an overview', 'author': 'Bro
```

Selecting the author of book with key `1`:

```
>>> V = mylib.values()
>>> d = V[int(mylib.keys()[1])]
>>> eval(d)['author']
'Brookshear'
```

DBM File Operations

an overview

Python code	description
`import dbm`	load module `dbm`
`f = dbm.open('n','c')`	create or open dbm file with name `n`
`f['key'] = 'value'`	assign `value` for `key`
`f.keys()`	returns the keys
`value = f['key']`	load `value` for `key`
`count = len(f)`	number of entries stored
`found = 'key' in f`	see if entry for `key`
`del f['key']`	remove entry for `key`
`f.close()`	close dbm file

Typical use:

- every record in database has unique key,
- values are dictionaries, stored as strings.

mass storage
dictionaries in Python

1. Files and Databases
   - mass storage
   - hash functions
2. Dictionaries
   - logical key values
   - nested tables
3. Persistent Data
   - storing information between executions
   - using DBM files
4. **Rule Based Programming**
   - **storing rules in dictionaries**
5. Summary + Assignments

Rule Based Programming: rules in dictionaries

Some rules for differentiation:

```
>>> D = {'sin(x)':'cos(x)' , \
...       'cos(x)':'-sin(x)'}
```

To prevent evaluation, the keys and values are strings.

Applying the rules = consulting the dictionary.

```
>>> D['sin(x)']
'cos(x)'
>>> D['cos(x)']
'-sin(x)'
```

Differentiation and Integration – with Sage and SymPy

For differentiation and integration (calculus),
Sage contains `Maxima` (Lisp) and `SymPy` (Python).
`SymPy` can be downloaded and installed separately.

Some examples:

```
sage: diff(cos(x),x)
-sin(x)
sage: integral(sin(x),x)
-cos(x)
```

Summary + Assignments

In this lecture we covered

- sections 1.2,1.3 in *Computer Science: an overview*
- more of chapter 4 of *Python Programming in Context*

Assignments:

1. For the computers in the lab, find the clock speed, the capacity of the internal memory and disk.
2. Use a dictionary to record state capitols.
3. Store the money exchange rates between dollar, euro, and yen in a dictionary and illustrate how to convert any sum of money.
4. Make a dictionary `I` to store the antiderivation rules for common trigonometric functions, sin, cos, and tan.
5. Give the Python commands to use `dbm` for storing the mileage tables of this lecture.