

Files and
Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent
Data

storing information
between executions
using DBM files

Rule Based
Programming

storing rules in
dictionaries

Summary +
Assignments

- 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

mass storage dictionaries in Python

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

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.

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.

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.

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.

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

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.

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

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.

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.

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.

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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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:

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

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:

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

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:

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

I/O Disk Operations

reading from and writing information to disk

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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:

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

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

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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

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

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

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

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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

Sequential File Organization

order records sequentially

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

Sequential File Organization

order records sequentially

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

Sequential File Organization

order records sequentially

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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.

Hash-based File Organization

order of records is computed

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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.

Hash-based File Organization

order of records is computed

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

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
```

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
```

Files and
Databasesmass storage
hash functions

Dictionaries

logical key values
nested tablesPersistent
Datastoring information
between executions
using DBM filesRule Based
Programmingstoring rules in
dictionariesSummary +
Assignments

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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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.

Operations on Dictionaries

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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}
```

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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}
```

Files and
Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent
Data

storing information
between executions
using DBM files

Rule Based
Programming

storing rules in
dictionaries

Summary +
Assignments

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}
```

Files and
Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent
Data

storing information
between executions
using DBM files

Rule Based
Programming

storing rules in
dictionaries

Summary +
Assignments

mass storage dictionaries in Python

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

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

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

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 `raw_input()` can immediately be used as 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 Wed 27 Jan 2010 : a mileage table
#
# A dictionary records the distance from Chicago
# to 3 other cities. The result of a raw_input
# is the key to query the dictionary.
#
Chicago = { 'Los Angeles' : 2047, 'Miami' : 1237, \
            'New York' : 807 }
city = raw_input('Give a city : ')
print 'Chicago -', city, ':', \
      Chicago[city], 'miles'
```

Mileage Tables

distances between cities

The program saved as [mileage.py](#):

```
# L-7 MCS 260 Wed 27 Jan 2010 : a mileage table
#
# A dictionary records the distance from Chicago
# to 3 other cities. The result of a raw_input
# is the key to query the dictionary.
#
Chicago = { 'Los Angeles' : 2047, 'Miami' : 1237, \
            'New York' : 807 }
city = raw_input('Give a city : ')
print 'Chicago -', city , ':' , \
      Chicago[city] , 'miles'
```

Nested Dictionaries

building 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
$
```

Nested Dictionaries

building 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
$
```

Nested Dictionaries

building 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
```

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
```

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent

Data

storing information
between executions
using DBM files

Rule Based

Programming

storing rules in
dictionaries

Summary +

Assignments

```
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 = raw_input('Give first city : ')
city2 = raw_input('Give second city : ')
print city1 , '-', city2 , ':' , \
      distance[city1][city2] , 'miles'
```

Files and
Databasesmass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent

Data

storing information
between executions
using DBM files

Rule Based

Programming

storing rules in
dictionaries

Summary +

Assignments

```
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 = raw_input('Give first city : ')
city2 = raw_input('Give second city : ')
print city1 , '-', city2 , ':' , \
      distance[city1][city2] , 'miles'
```

mass storage dictionaries in Python

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

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.

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.

Files and
Databasesmass storage
hash functions

Dictionaries

logical key values
nested tablesPersistent
Datastoring information
between executions
using DBM filesRule Based
Programmingstoring rules in
dictionariesSummary +
Assignments

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.

Files and
Databasesmass storage
hash functions

Dictionaries

logical key values
nested tablesPersistent
Datastoring information
between executions
using DBM filesRule Based
Programmingstoring rules in
dictionariesSummary +
Assignments

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.

Files and
Databasesmass storage
hash functions

Dictionaries

logical key values
nested tablesPersistent
Datastoring information
between executions
using DBM filesRule Based
Programmingstoring rules in
dictionariesSummary +
Assignments

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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using **DBM files**

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

Using DBM Files

DBM files are standard in the Python library.

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

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

```
>>> 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': 'Mi
$ ls
```

→ library is a file in current directory.

Using DBM Files

DBM files are standard in the Python library.

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

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

```
>>> 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': 'Mi
$ ls
```

→ library is a file in current directory.

Using DBM Files

DBM files are standard in the Python library.

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

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

```
>>> 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': 'Mi
$ ls
```

→ library is a file in current directory.

Using DBM Files

DBM files are standard in the Python library.

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

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

```
>>> 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': 'Mi
$ ls
```

→ library is a file in current directory.

Adding Books to the Library

and selecting books using the key

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Selecting the author of book with key 1:

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

Adding Books to the Library

and selecting books using the key

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Selecting the author of book with key 1:

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

Adding Books to the Library

and selecting books using the key

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Selecting the author of book with key 1:

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

Adding Books to the Library

and selecting books using the key

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Selecting the author of book with key 1:

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

Adding Books to the Library

and selecting books using the key

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

DBM File Operations

an overview

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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

Typical use:

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

mass storage dictionaries in Python

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

- 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

storing rules in dictionaries

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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)'
```

Rule Based Programming

storing rules in dictionaries

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

Summary + Assignments

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)'
```

Files and
Databasesmass storage
hash functions

Dictionaries

logical key values
nested tablesPersistent
Datastoring information
between executions
using DBM filesRule Based
Programmingstoring rules in
dictionariesSummary +
Assignments

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)
```

Files and
Databasesmass storage
hash functions

Dictionaries

logical key values
nested tablesPersistent
Datastoring information
between executions
using DBM filesRule Based
Programmingstoring rules in
dictionariesSummary +
Assignments

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

Files and Databases

mass storage
hash functions

Dictionaries

logical key values
nested tables

Persistent Data

storing information
between executions
using DBM files

Rule Based Programming

storing rules in
dictionaries

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 in Python.
- 4 Make a dictionary \mathbb{I} to store the antiderivation rules for common trigonometric functions, sin, cos, and tan.
- 5 Give the Python commands to use `anydbm` for storing the mileage tables of this lecture.