SEE1002
Introduction to Computing for Energy and Environment

Part 3: Basic Python programming

Sec. 4: Derived data structures

Course Outline

Part 1: Introduction to computing

Part 2: Elements of Python programming

Section 1: Fundamentals

Section 2: Branching or Decision Making

Section 3: Loops

Section 4: Functions

Part 3: Basic Python programming

Section 1: Modules

Section 2: Structure of a Python program

Section 3: Good programming practices

Section 4: Derived data structures

Part 4: Python for science and engineering

Section 1: Vectors, matrices and arrays

Section 2: Useful mathematical and statistical functions

Section 3: Plotting

Section 4: Input and output

Part 5: Applications

Introduction

Importance of data structures

- We have talked about data structures from time to time.
- They are important because for most of the problems that you'll be working on, you will need to work with data.
- An appropriate choice of data structure simplifies the problem.

Derived data structures

- We have already talked about lists, tuples and dictionaries.
- In the final section we will discuss a data structure known as an array, which is appropriate for vectors and fields.
- More generally, however, it's useful to be able to create custom or derived data structures from basic data structures that we have already learned.

Why do we need custom data structures?

- Some data doesn't naturally fit into standard data structures.
- However, we can **create new data structures.**
- This is an important topic in computer science but we won't discuss it in detail.
- We're going to cover simple applications of what we have already learned.

1. List of lists

Combining lists

- A list can be created using arbitrary elements.
- Thus the elements of a list can also be lists!
- Examples:
  - `list1d=[1,2,3],`
  - `list2d=[ [1,1], [2,2], [3,3] ]`
  - `etc.`

Access

- A list of lists can be accessed using separate indices for each list. They can be referred to as

```
listoflists[i][j]…[m]
```

where `i` is the index for the outermost list and `m` is the index for the innermost list.

Order is crucial!

- The outermost list refers to the outer pair of brackets!

Innermost (i.e. like list inside a list element)

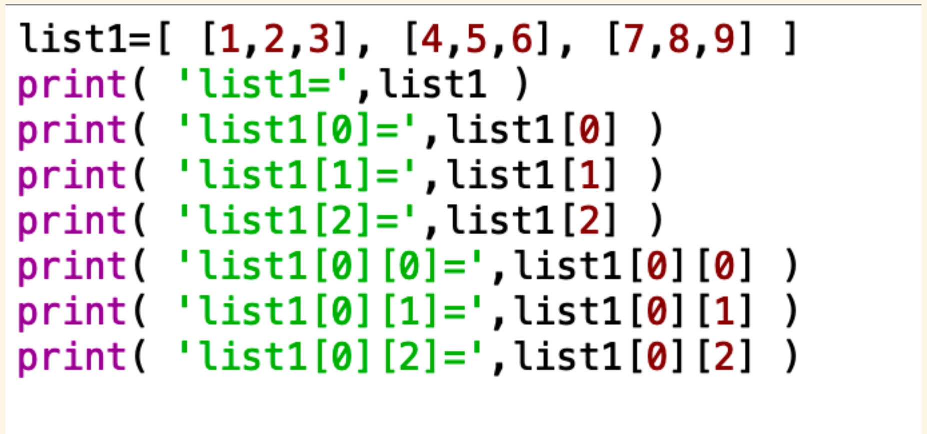
- `list2d=[ [1,2,3], [4,5,6], [7,8,9] ]`

Outermost (i.e. like regular 1-D list)

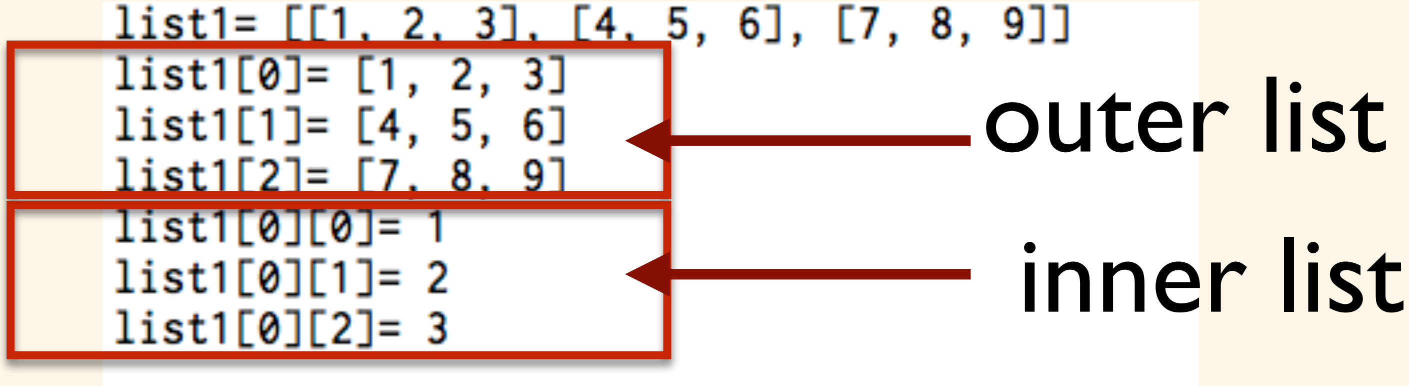
Why do we follow this order?

- In theory, we could do things the other way around!
- This is explained in Sec. 4.2. Basically this is a convention. In most computer programming languages (e.g. Python), the final index (i.e. the column) varies more rapidly.

Example 1: A 2-D list of lists



```
list1=[ [1,2,3], [4,5,6], [7,8,9] ]
print( 'list1=',list1 )
print( 'list1[0]=',list1[0] )
print( 'list1[1]=',list1[1] )
print( 'list1[2]=',list1[2] )
print( 'list1[0][0]=',list1[0][0] )
print( 'list1[0][1]=',list1[0][1] )
print( 'list1[0][2]=',list1[0][2] )
```



```
list1= [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
list1[0]= [1, 2, 3]
list1[1]= [4, 5, 6]
list1[2]= [7, 8, 9]
list1[0][0]= 1
list1[0][1]= 2
list1[0][2]= 3
```

outer list

inner list

Applications

- A list of lists is a convenient way of combining data.
- Instead of having a bunch of different lists, we can have a single list. This makes the program more compact and easier to read.
- Examples
  - Grades for each student in a class
  - Data at different sites

2. List of dictionaries

Combining dictionaries

- This is useful for providing detailed information about a list of items, each of which has certain properties or features.

- Example (a list of students):

  - `student0 ={‘sid’:12345678, ‘grades’:[‘A’,’A’,’A’,’A’,'A'], ‘phone’:11111111, ‘surname’:’Lee’}`

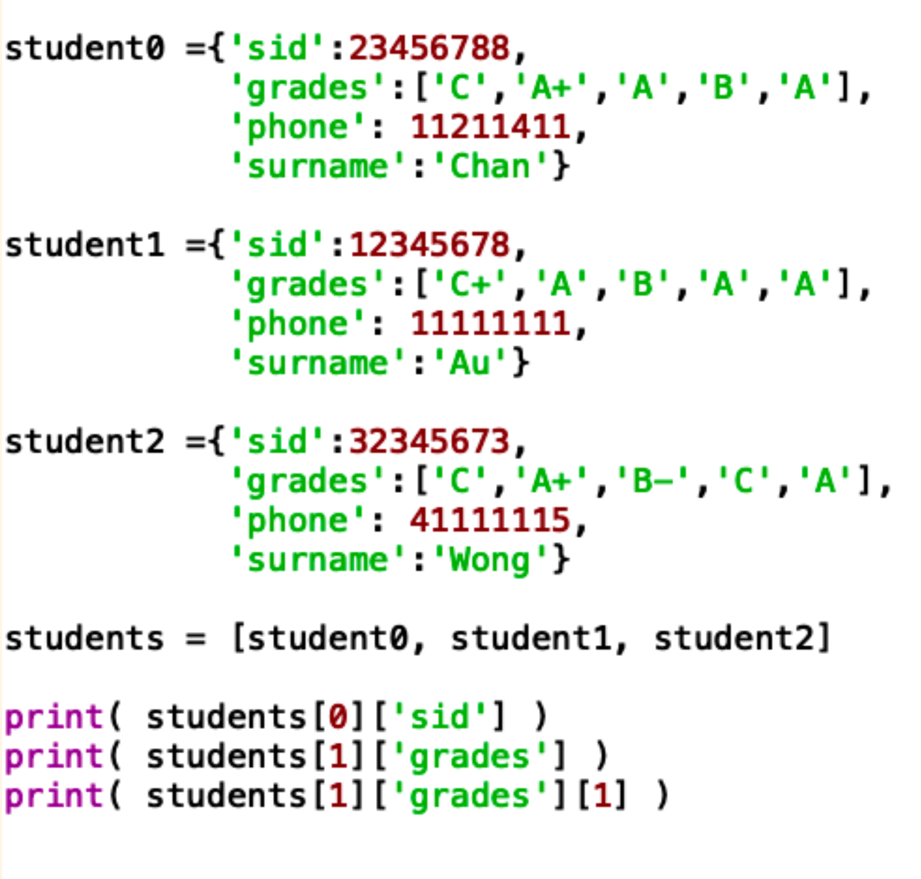
  - `student1 = {…}`

  - `student2 = {…}`

N.B. This is the correct order

- **`students=[ student0, student1, student2 ]`**

Example 2: A list of dictionaries



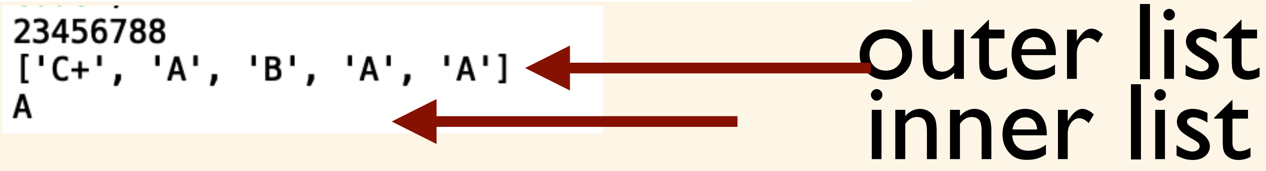
```
student0 ={'sid':23456788,
           'grades':['C','A+','A','B','A'],
           'phone': 11211411,
           'surname':'Chan'}

student1 ={'sid':12345678,
           'grades':['C+','A','B','A','A'],
           'phone': 11111111,
           'surname':'Au'}

student2 ={'sid':32345673,
           'grades':['C','A+','B-','C','A'],
           'phone': 41111115,
           'surname':'Wong'}

students = [student0, student1, student2]

print( students[0]['sid'] )
print( students[1]['grades'] )
print( students[1]['grades'][1] )
```



```
23456788
['C+', 'A', 'B', 'A', 'A']
A
```

outer list

inner list

Interpretation

- Each item of the list corresponds to a different student.
- For each student, we have information about various properties (e.g. student id, grades).

3. Dictionary of lists

Combining lists in an intuitive way

- This is useful for providing detailed information about different properties.
- Example (dictionary listing student info):

```
• class={ ‘sid’:list1,
         ’grades’: list2,
         ‘telephone’:list3
       }
```

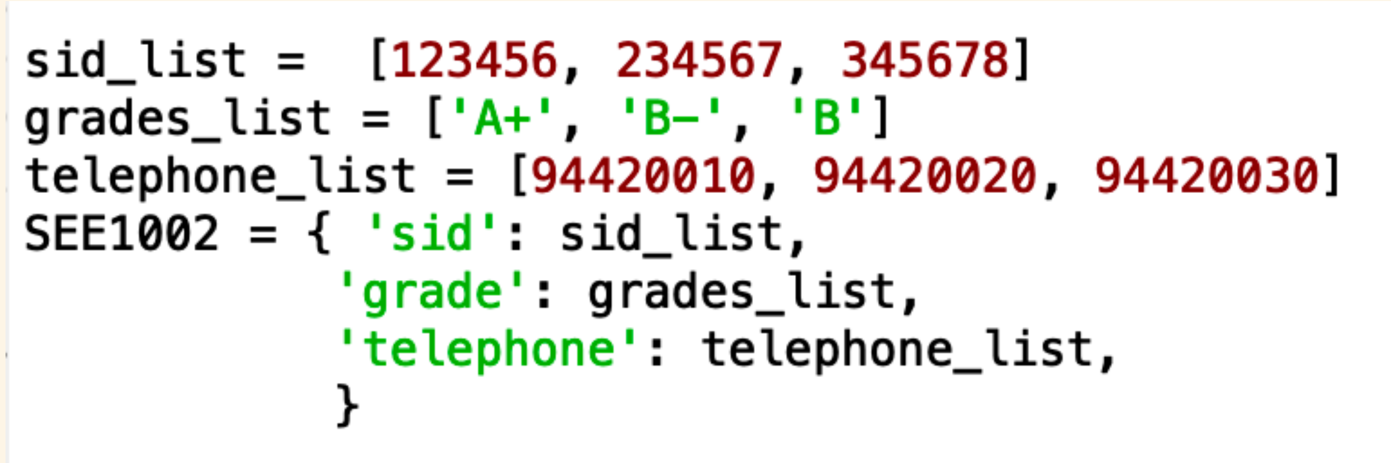
N.B. The order has been swapped for clarity!

```
‣ list1 =[12345678, 23456789, 34567890, …]
‣ list2 =[‘A+’, ‘B’, ‘C-‘, …]
‣ list3 =[34415678, 34436789, 34405678, …]
```

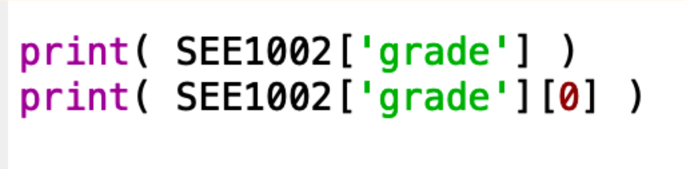
Interpretation

- Each dictionary key corresponds to a different property.
- For each property, we have information about all the students.

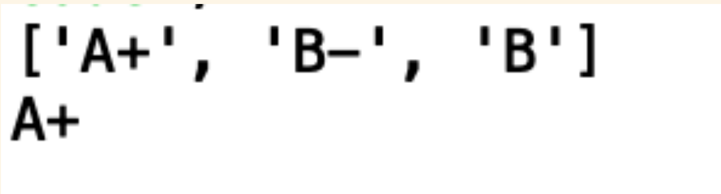
Example 3: A dictionary of lists



```
sid_list =  [123456, 234567, 345678]
grades_list = ['A+', 'B-', 'B']
telephone_list = [94420010, 94420020, 94420030]
SEE1002 = { 'sid': sid_list,
           'grade': grades_list,
           'telephone': telephone_list,
           }
```



```
print( SEE1002['grade'] )
print( SEE1002['grade'][0] )
```



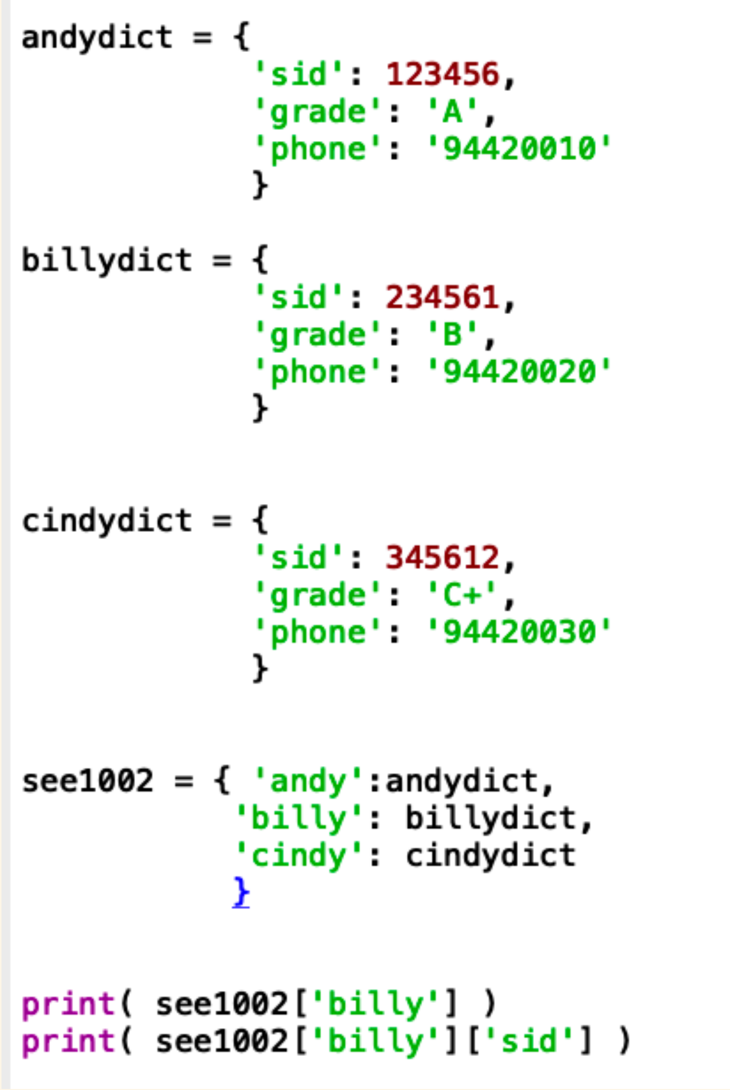
```
['A+', 'B-', 'B']
A+
```

4. Dictionary of dictionaries

Combining dictionaries in an intuitive way

- This is useful for providing information about <u>different properties of labelled items</u>.
- Example (a student dictionary):
  - `andydict={‘sid’:12345, eid=‘abclau’}`
  - `billydict = {…}`
  - `cindydict = {…}`
  -
    ```
    student={’andy’: andydict,
             ‘billy’: billydict,
             ‘cindy’: cindydict }
    ```

Example 4: A dictionary of dictionaries



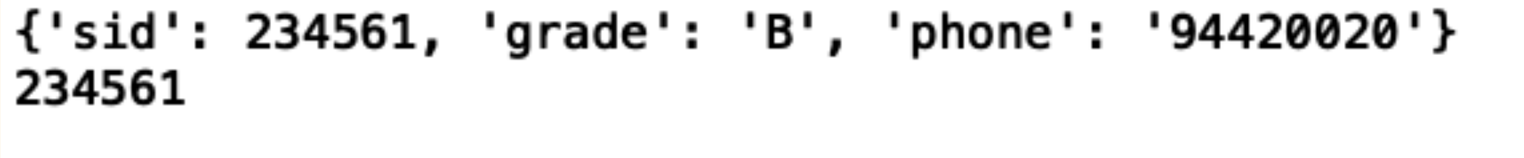
```
andydict = {
            'sid': 123456,
            'grade': 'A',
            'phone': '94420010'
            }

billydict = {
            'sid': 234561,
            'grade': 'B',
            'phone': '94420020'
            }


cindydict = {
            'sid': 345612,
            'grade': 'C+',
            'phone': '94420030'
            }


see1002 = { 'andy':andydict,
           'billy': billydict,
           'cindy': cindydict
           }


print( see1002['billy'] )
print( see1002['billy']['sid'] )
```



```
{'sid': 234561, 'grade': 'B', 'phone': '94420020'}
234561
```

Summary

1. Derived data structures make it easier to work with complicated datasets.