Accelerating Information Technology

http://aiti.mit.edu

Ghana Summer 2012
Lecture 6 – Data Structures

MISTI

Lists

Lists

- List is a sequence of values
- String is a sequence of characters
  - ‘banana’

- List can be a sequence of any type
  - [10, 20, 30, 40] - integers
  - ['crunchy frog', 'ram bladder', 'lark vomit'] - strings
  - ['spam', 2.0, 5, [10, 20]] - all mixed!!

Creating a list

- Empty list
  - empty_list = []

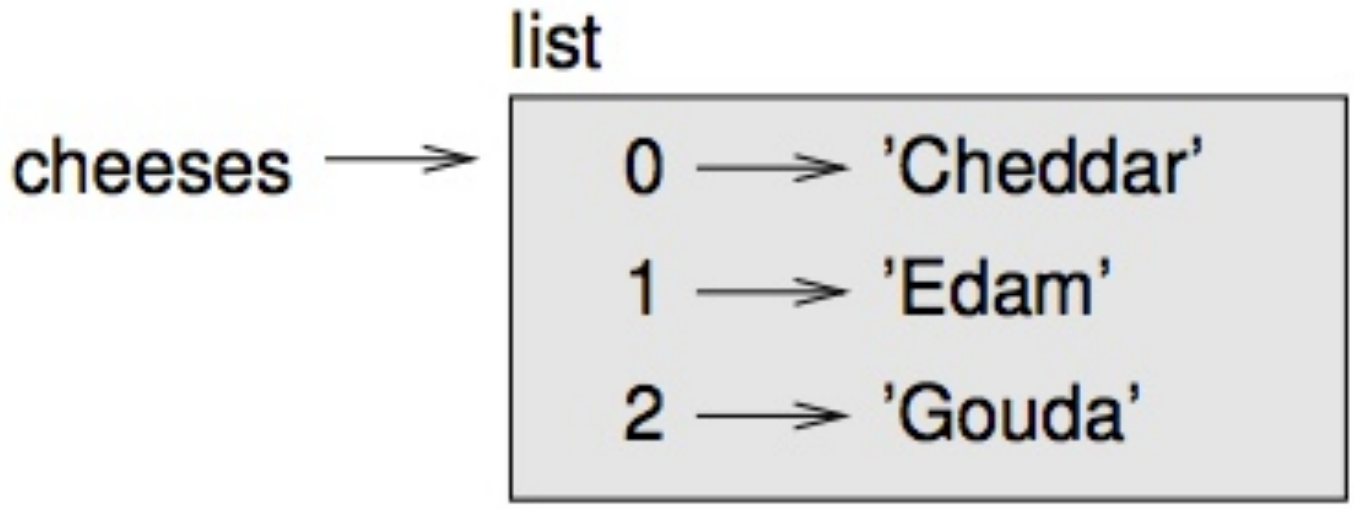
- >>> cheeses = ['Cheddar', 'Edam', 'Gouda']
- >>> numbers = [17, 123]
- >>> empty = []
- >>> print cheeses, numbers, empty
- ['Cheddar', 'Edam', 'Gouda'] [17, 123] []

Lists

- Visualize lists like a collection of numbered buckets
- Indexing starts at 0

Indexing

- cheeses = ['Cheddar', 'Edam', 'Gouda']



- >>> print cheeses[0]
- >>> Cheddar

Lists are mutable

- mutable = we can change their values
- Example:

```
>>> numbers = [17, 123]
>>> print numbers[1]
>>> 123
>>> numbers[1] = 5
>>> print numbers
[17, 5]
```

Lists are mutable

- mutable = we can change their values
- But be careful!

- Example:

```
>>> numbers = [17, 123]
>>> print numbers[2]
>>> IndexError: list index out of range
```

Lists - useful operation

- You can check whether element is in the list

```
>>> cheeses = ['Cheddar', 'Edam', 'Gouda']
>>> 'Edam' in cheeses
True
>>> 'Brie' in cheeses
False
```

Lists-useful operations

- The + operator concatenates lists:

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print c
[1, 2, 3, 4, 5, 6]
```

Slice operators

- what if we want to get part of the list or string?
- Use slice operators!

```
>>> s = 'Monty Python'
>>> print s[0:5]
Monty
>>> print s[6:12]
Python
```

Slice operators on lists

- >>> t = ['a', 'b', 'c', 'd', 'e', 'f']
  >>> t[1:3]
  ['b', 'c']
  >>> t[:4]
  ['a', 'b', 'c', 'd']
  >>> t[3:]
  ['d', 'e', 'f']

Slice operators on lists

- >>> t = ['a', 'b', 'c', 'd', 'e', 'f']
  >>> t[1:3] = ['x', 'y']
  >>> print t
  ['a', 'x', 'y', 'd', 'e', 'f']

List methods

- *append* - adds new element to the end

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t
['a', 'b', 'c', 'd']
```

List methods

- *sort* - arranges the elements of the list from low to high

```
>>> t = ['d', 'c', 'e', 'b', 'a']
>>> t.sort()
>>> print t
['a', 'b', 'c', 'd', 'e']
```

List methods

- *insert* - inserts an item at a given position

```
>>> t = ['banana', 'mango', 'coconut']
>>> t.insert(2, 'watermelon')
>>> print t
['banana', 'mango', 'watermelon', 'coconut']
```

List methods

- *remove*- removes the first item with a given value

```
>>> t = ['banana', 'mango', 'coconut']
>>> t.remove('mango')
>>> print t
['banana', 'coconut']
```

Lists: Iteration

- How can we print out all elements of the list, using a few lines of code?

  t = ['banana', 'mango', 'coconut']

  - Iteration over the items in the list

```
for fruit in t:
    print fruit
```

  - Iteration over indices

```
for index in range(len(t)):
    print t[index]
```

Tuples: Introduction

- Essentially an **immutable** list
  - **CANNOT** change list items
  - Form: `tuple=('a', 'b', 'c', 'd',…)`
  - `fruits_tuple = ('banana', 'mango', 'coconut')`

Tuples: Manipulation

- **NOTICE:**
  - `tuple[0] = 'A'` returns an error

- There are some ways around this
  - Make new tuple and add part of existing tuple
  - `tuple = ('A',) + tuple[1:]`
  - New Tuple: `('A', 'b', 'c', 'd', 'e')`

Lists and Tuples: Limitations

- aiti_students = [‘TK’, ‘Priscilla’, ‘Gifty’, ‘Selom’, ...]
- UGL_students - all ~40000 students that go to UGL

- What if I wanted to check which one of you goes to UGL?
- I would have to go through 40000 names!

Lists and Tuples: Limitations

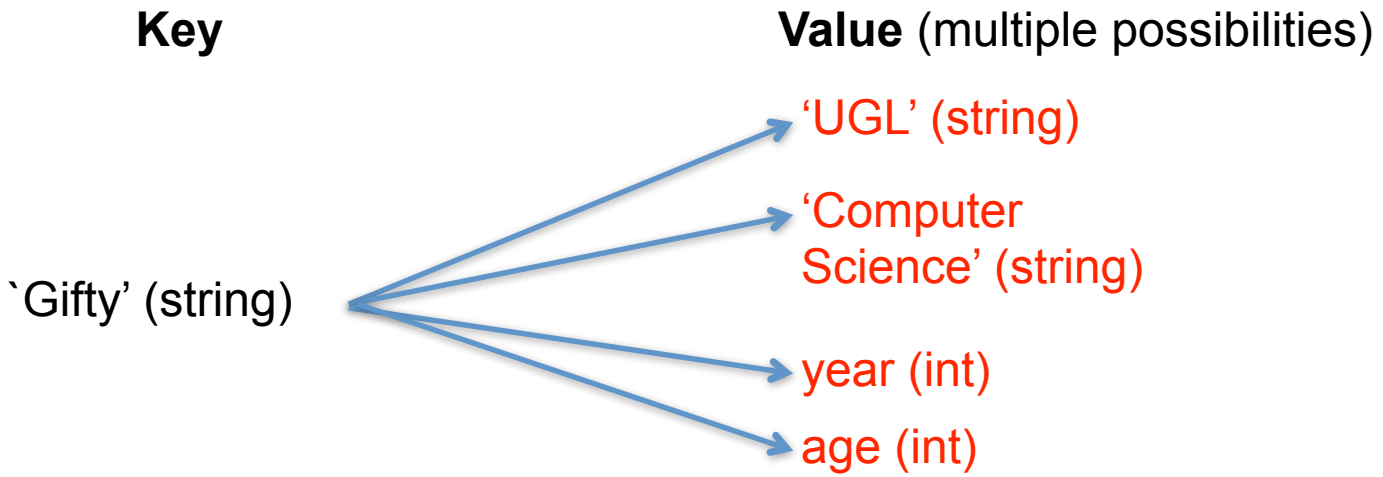
- What if I wanted to check which one of you goes to UGL?
- We would have to go through 40000 names!
- Are there any shortcuts?
  - Sorted lists can help
  - Costly to insert new elements into sorted lists

- A different solution: **dictionaries**

Dictionaries

- An unordered collection of (key,value) pairs
- (key, value) pairs are mappings
  - key: something you know
  - value: something you want to know that is related to the key
- Key and value can be objects of any type

Text



Dictionaries: Initialization

- Initialization (maps students to years):

```
aiti_students = {'Darko':'UGL',
'Mayi':'Kwame',
'Ernest':'GIMPA'}
```

Key	Value
Darko	UGL
Mayi	Kwame
Ernest	GIMPA

Dictionaries: Modification

- Modification
  - Change Darko's school:
    `aiti_students['Darko']='MIT'`

Key	Value
Darko	MIT
Mayi	Kwame
Ernest	GIMPA

Dictionaries: Modification

- Modification:
  - Add a new student:
    `aiti_students['Gifty']='UGL'`

Key	Value
Darko	UGL
Mayi	Kwame
Ernest	GIMPA
Gifty	UGL

Dictionaries

- Suppose someone gives you a list of students, `aiti_student_list`
- How can we use our dictionary, `aiti_students`, to print out the teams of each player on the `aiti_students_list`?
- We may not know that `aiti_students` has an entry for an item in `aiti_students_list`
- ```
  for student in aiti_student_list:
     if student in aiti_student:
        print aiti_student[student]
     else:
        print 'unknown school'
  ```
- Later on: exception handling
  ```

Useful Questions

- Will one set of data be mapped to another?
  - Words to definitions, soccer players to jersey sizes, students to grades
  - Dictionary!

Useful Questions

- Is the data I'm storing going to change?
  - Mutability VS Immutability
  - If NOT → *Tuples!*

- If data will change? Can it fit into a single list?
  - If YES → *Use a List!*
  - Recall it has: *add*, *remove* and *sort* methods