Week 7

Video 1: Tuples

Immutable Sequences

Tuples are immutable sequences: they cannot be modified. Tuples and lists have much in common, but lists are mutable sequences: they can be modified.

Tuples use parentheses instead of square brackets:

```
lst = ['a', 3, -0.2]
tup = ('a', 3, -0.2)
```

Once created, items in lists and tuples are accessed using the same notation:

```
>>> lst[0]
'a'
>>> tup[0]
'a'
```

Slicing can be used with both:

```
>>> lst[:2]
['a', 3]
>>> tup[:2]
('a', 3)
```

Tuples cannot be modified:

```
>>> tup[0] = 'b'
TypeError: 'tuple' object does not support item assignment
```

Tuples have fewer methods than lists. In fact, the only regular methods are `count` and `index`:

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__',
'__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'count', extend', 'index', 'insert', 'pop', 'remove', 'reverse', sort']
>>> dir(tuple)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__', '__iter__',
'__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'count', 'index']
```

The rest of the list methods are not available for tuple because they modify the object, and tuples, being immutable, cannot be modified. A `for` can be used to iterate over the values in a tuple:

```
>>> tup = ('a', 3, -0.2)
>>> for item in tup:
    print(item)

a
3
-0.2
```

A tuple can be passed as an argument to the built-in function `len`:

```
>>> len(tup)
3
```

It is also possible to iterate over the indices of a tuple:

```
>>> for i in range(len(tup)):
        print(tup[i])

a
3
-0.2
```

Optional reading

Chapter 11.2 Storing Data Using Tuples

Video 2: Type dict

Dictionary

Another way to store collections of data is using Python's dictionary type: `dict`.

The general form of a dictionary is:

```
{key1: value1, key2: value2, ..., keyN: valueN}
```

Keys must be unique. Values may be duplicated. For example:

```
asn_to_grade = {'A1': 80, 'A2': 90, 'A3': 90}
```

In the example above, the keys are unique: 'A1', 'A2' and 'A3'. The values are not unique: 80, 90 and 90.

How To Modify Dictionaries

Dictionaries are mutable: they can be modified. There are a series of operations and methods you can apply to dictionaries which are outlined below.

Operation	Description	Example

object in dict	Checks whether `object` is a key in `dict`.	>>> asn_to_grade = {'A1': 80, 'A2': 90, 'A3': 90}
		>>> 'A1' in asn_to_grade
		True
		>>> 80 in asn_to_grade
		False

len(dict)	Returns the number of keys in `dict`.	>>> asn_to_grade = {'A1': 80, 'A2': 90, 'A3': 90}
		>>> len(asn_to_grade)
		3

del dict[key]	Removes a key and its associated value from `dict`.	>>> asn_to_grade = {'A1': 80, 'A2': 90, 'A3': 90}
		>>> del asn_to_grade['A1']
		>>> asn_to_grade
		{'A3': 90, 'A2': 90}

dict[key] = value	If `key` does not exist in `dict`, adds key and its associated `value` to `dict`. If `key` exists in `dict`, updates `dict` by setting the value associated with `key`	>>> asn_to_grade = {'A1' : 80, 'A2': 90, 'A3' : 90} >>> asn_to_grade['A4'] = 70

value	to `value`.	>>> asn_to_grade {'A1': 80, 'A3': 90, 'A2': 90, 'A4': 70}

Accessing Information From Dictionaries

Dictionaries are unordered. That is, the order the key-value pairs are added to the dictionary has no effect on the order in which they are accessed. For example:

```
>>> asn_to_grade = {'A1': 80, 'A2': 70, 'A3': 90}
>>> for assignment in asn_to_grade:
    print(assignment)

A1
A3
A2
```

The for-loop above printed out the keys of the dictionary. It is also possible to print out the values:

```
>>> asn_to_grade = {'A1': 80, 'A2': 70, 'A3': 90}
>>> for assignment in asn_to_grade:
    print(asn_to_grade[assignment])

80
90
70
```

Finally, both the keys are values can be printed:

```
>>> asn_to_grade = {'A1': 80, 'A2': 70, 'A3': 90}
>>> for assignment in asn_to_grade:
    print(assignment, asn_to_grade[assignment])

A1 80
A3 90
A2 70
```

Empty Dictionaries

A dictionary can be empty. For example:

```
d = {}
```

Heterogeneous Dictionaries

A dictionary can have keys of different types. For example, one key can be of type `int` and another of type `str`:

```
d = {'apple': 1, 3: 4}
```

Immutable Keys

The keys of a dictionary must be immutable. Therefore, lists, dictionary and other mutable types cannot be used as keys. The following results in an error:

```
d[[1, 2]] = 'banana'
```

Since lists are mutable, they cannot be keys. Instead, to use a sequence as a key, type tuple can be used:

```
d[(1, 2)] = 'banana'
```

Optional reading

Chapter 11.3 Storing Data Using Dictionaries

Video 3: Inverting a Dictionary

Switching Keys and Values

Dictionaries have keys that are unique and each key has a value associated with it. For example, here is a dictionary mapping fruit to their colours:

```
fruit_to_colour = {'watermelon': 'green', 'pomegranate': 'red',
'peach': 'orange', 'cherry': 'red', 'pear': 'green',
'banana': 'yellow', 'plum': 'purple', 'orange': 'orange'}
```

To invert the dictionary, that is, switch the mapping to be colours to fruit, here is one approach:

```
>>> colour_to_fruit = {}
>>> for fruit in fruit_to_colour:
    colour = fruit_to_colour[fruit]
    colour_to_fruit[colour] = fruit



>>> colour_to_fruit
{'orange': 'orange', 'purple': 'plum', 'green': 'pear', 'yellow': 'banana', 'red': 'pomegranate'}
```

The resulting dictionary is missing some fruit. This happens since colours, which are keys, are unique so later assignments using the same colour replace earlier entries. A way to remedy this is to map colours to a list of fruit.

Mapping A Key To A List

For the example above, we need to consider two cases when adding a colour and a fruit to the dictionary:

1. If the colour is not a key in the dictionary, add it with its value being a single element a list consisting of the fruit.
2. If the colour is already a key, append the fruit to the list of fruit associated with that key.

.

```
>>> colour_to_fruit = {}
>>> for fruit in fruit_to_colour:
    # What colour is the fruit?
    colour = fruit_to_colour[fruit]
    if not (colour in colour_to_fruit):
        colour_to_fruit[colour] = [fruit]
    else:
        colour_to_fruit[colour].append(fruit)

>>> colour_to_fruit
{'orange': ['peach', 'orange'], 'purple': ['plum'], 'green': ['watermelon', 'pear'],     'yellow': ['banana'], 'red': ['cherry', '
```

Optional reading

Chapter 11.4 Inverting a Dictionary

Video 4: Populatingh a dictionary

No notes on Coursera (9/30/2013 8:43:48 PM GMT+3)