Tuples, Dictionaries and Sets

(3 new data types in Python)
Tuples, Dictionaries and Sets
(3 new data types in Python)

*Think, Python* chapters on dictionaries and tuples are *very* good.
Tuples
A tuple is a comma-separated sequence of objects of any type.

x=3,5,7
y = ‘computers’,47,[‘a’,’b’]

How do you make an empty tuple? A tuple with only one element?
Tuples are sequences

Like strings, and lists, and bytes objects

So all the stuff like indexing, slices, `len` function, `+` for concatenation, `*` for iteration, traversing with `for` and `in`, can be used with tuples.
Parentheses are (usually) not necessary.

It’s the commas that make it a tuple, not the parentheses.

**Examples.** What do you get from

\[
x=(3,4),5,6
\]

- `type(x)`
- `len(x)`
- `3,4+5,6`
- `3,4+(5,6)`
- `(3,4)+(5,6)`
Tuples are immutable (like strings, unlike lists)

\[ x=(4,1,9,2) \]
\[ x[2]=3 \quad \text{you can’t do this!} \]
\[ x=x[:2]+(3,)+x[3:] \]

See the function tuple_abcs() in the demos.
Functions returning tuples

Example: Built-in function `divmod`, quadratic equation example.

See the function `solve_quadratic` in the code demos.
Changing a pair of variables with tuple assignment.

Example. Fibonacci sequence:

0,1,1,2,3,5,8,13,21,... Each term is sum of the previous two. To compute it, you have to keep track of both the most last two values, and repeatedly execute:

\((\text{previous}, \text{current})\)\(\rightarrow\) \((\text{current}, \text{current}+\text{previous})\)

e.g., \((0,1)\rightarrow (1,1)\rightarrow (1,2)\rightarrow (2,3)\rightarrow (3,5)\)....

Tuples provide a quick solution.

See the function fibonacci in the code demos.
Wait a second.....

Could have done these examples with lists.

Furthermore, you can do *more* with lists, because they’re mutable.

So why bother with tuples?
Lists cannot be used as keys for a Python dictionary (next topic!)

Functions with a variable number of arguments

Examples: print(x), print(x, 'hello', y+z), max(2, 3), max(a, b, c, d), ...

Defined as def f(*x): In the function, x is a tuple consisting of the sequence of arguments passed to f.

See the function product in the code demos.
Dictionaries
## Dictionaries (abstractly)

<table>
<thead>
<tr>
<th></th>
<th>List (of tuples)</th>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;New York&quot;, 8398748</td>
<td>&quot;New York&quot; 8398748</td>
</tr>
<tr>
<td>1</td>
<td>&quot;Los Angeles&quot;, 3990456</td>
<td>&quot;Los Angeles&quot; 3990456</td>
</tr>
<tr>
<td>2</td>
<td>&quot;Chicago&quot;, 2705994</td>
<td>&quot;Chicago&quot; 2705994</td>
</tr>
<tr>
<td>3</td>
<td>&quot;Houston&quot;, 2345502</td>
<td>&quot;Houston&quot; 2345502</td>
</tr>
<tr>
<td>4</td>
<td>&quot;Phoenix&quot;, 1660372</td>
<td>&quot;Phoenix&quot; 1660372</td>
</tr>
</tbody>
</table>
Dictionaries (in Python)

d={'New York': 8398748, 'Los Angeles': 3990456, 'Chicago': 2705994, 'Houston': 2345502, 'Phoenix': 1660272}

Show the demo
Dictionaries

- A dictionary is a collection of key-value pairs:
- Kind of like a list: `len` function, `for... in`, mutable
- But...items are indexed by keys, not integers, and `x in d` asks whether `x` is a key in `d`, not a value.
- Any type of object at all can be used as a value. Keys are restricted: strings, ints, floats, tuples of objects that can be used as keys; but not lists, not dicts.
Example. We want to work with a collection of films and their casts.
**Example.** Simplest and most efficient handling is to represent the information in a dictionary whose keys are film titles and values are lists of actors.

```
{'Ford vs. Ferrari': ['Christian Bale', 'Tracy Letts'],
 'Little Women': ['Saoirse Ronan', 'Laura Dern', 'Meryl Streep', 'Tracy Letts'], ...
```
Example. What if we wanted to do a ‘reverse lookup’? Given an actor, find the movies that they’re in.

Ford vs. Ferrari
- Christian Bale

Little Women
- Tracy Letts
- Saoirse Ronan

Ladybird
- Laura Dern

Marriage Story
- Meryl Streep
- Scarlett Johansson
- Adam Driver

{’Tracy Letts’:[’Ladybird’,’Little Women’,’Ford vs. Ferrari’],’Laura Dern’:[’Little Women’,’Marriage Story’],’Adam Driver’:[’Marriage Story’]… }

See function invert_dict in code demos
**Example.** Tabulate the frequencies of words in a long text.

So, for example, the results might look like

‘the’:15682
‘and’:12816
‘to’:10308,

etc.
List of words in text

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>“THE”</td>
</tr>
<tr>
<td>1</td>
<td>“CAT”</td>
</tr>
<tr>
<td>2</td>
<td>“AND”</td>
</tr>
<tr>
<td>3</td>
<td>“THE”</td>
</tr>
<tr>
<td>4</td>
<td>“DOG”</td>
</tr>
<tr>
<td>5</td>
<td>“AND”</td>
</tr>
</tbody>
</table>

Dictionary of word-frequency pairs

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>“THE”</td>
<td>2</td>
</tr>
<tr>
<td>“CAT”</td>
<td>1</td>
</tr>
<tr>
<td>“AND”</td>
<td>2</td>
</tr>
<tr>
<td>“DOG”</td>
<td>1</td>
</tr>
<tr>
<td>Word</td>
<td>Frequency</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>&quot;THE&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;CAT&quot;</td>
<td>1</td>
</tr>
<tr>
<td>&quot;AND&quot;</td>
<td>2</td>
</tr>
<tr>
<td>&quot;DOG&quot;</td>
<td>1</td>
</tr>
</tbody>
</table>

Dictionary of word-frequency pairs

See functions in code demos that carry out and plot this tabulation
What’s going on:

With *lists*, when Python evaluates an expression like `L[3]`, it uses information about `L` together with the index 3 to perform a calculation determining where in the computer’s memory the value `L[3]` is stored. The calculation works the same whether it’s `L[3]` or `x[30000000]`, and the time to perform this does not depend on the size of the list. (Compare this with searching for a target value in a list.)
What’s going on:

With *dictionaries*, when Python evaluates an expression like `d[‘New York’]`, it uses information about `d` together with a special *hash value* calculated from the string ‘New York’, to determine where in the computer’s memory `d[‘New York’]` is stored. The time required to perform this calculation does not depend on the size of the dictionary.
What’s going on:

With *dictionaries*, when Python evaluates an expression like `d[‘New York’]`, it uses information about `d` together with a special *hash value* calculated from the string ‘New York’, to determine where in the computer’s memory `d[‘New York’]` is stored. The time required to perform this calculation does not depend on the size of the dictionary.

Mutable objects do not have associated hash values, and cannot be used as dictionary keys.

**Demo of hash function:** Just type `hash(x)` for various choices of objects `x`—try ints, strings, lists, tuples of ints, tuples of lists,….
Keys cannot be repeated in dictionaries

d={'a':2,'b':3,'a':5}

d
{"a": 5, 'b': 3}

Sets: Like dictionaries with only keys, not values