Collections: Lists, Dictionaries, and Tuples

Info 206

Niall Keleher

05 September 2017

Today's Quiz: http://bit.ly/2eCRLaK

Today's Outline

1. Lists
2. Dictionaries
3. Mutability
4. Group Projects - Initial team planning

Lists

- Ordered collections of arbitrary objects
- Allow for indexing, slicing, concatenation
- Variable in length, heterogenous, nestable
- Mutable sequence
- Foundation for arrays (nested sublists == multidimensional arrays)
- List comprehensions - building new lists or iterations

```
letters = ['N', 'i', 'a', 'l', 'l']

firstname = ''.join(letters)

print(firstname)
```

```
## Niall
```

Dictionaries

- ordered collections of objects - hash tables
- Accessed by key, not position
- Useful for attaching records to an identifier or for search tables
- Variable in length, heterogeneous, and nestable
- Mutable mapping

Dictionaries

```
first_dictionary = {1:"Niall", 2:"Eve"}

print(first_dictionary)

first_dictionary.update({3:"Paul"})

print(first_dictionary)
```

Dictionaries - keys must be unique

```
second_dictionary = {1:"Niall", 1:"Eve"}

print(second_dictionary)
```

Dictionary operations

```
my_dict = {'unit1':100, 'unit2':-54, 'unit3':247}

print(sum(my_dict.values()))
```

Lists vs. Dictionaries

- Need to access by position? Lists
- Unlabelled items? Lists
- Symbolic records? Dictionaries
- Labelled components? Dictionaries

Mutability

Mutability

- Changes can be made *in place*
- Mutable examples: lists, dictionaries, sets

Mutability - Sorting a list

```
cities = [Tokyo, Delhi, Jakarta, Chicago, Nairobi]

cities.sort()

sorted(cities)
```

Mutability - inefficient code

```
string_build = ""
for data in mylist:
    string_build += str(data)
```

Mutability - more efficient methods

```
### Method 1: for loop
aggregate_list = []
for data in mylist:
    aggregate_list.append(str(data))
"".join(aggregate_list)

### Method 2: list comprehension
"".join([str(data) for data in mylist])

e### Method 3: map function
"".join(map(str, mylist))
```

Mutability

```
a_list = ["tu", "tu"]

b_list = [a_list]

c_list = [a_list]

b_list[0][1] = "ba"

print("c_list now has the value", c_list)
```

Tuples

- Ordered collections of arbitrary objects
- Indexing, slicing, concatenating
- Similar to lists, share many of the same properties
- Immutable

Tuples vs. Lists

- Fixed objects - immutable
- Searching within tuples is more straightforward
- Tend to be faster
- Code can be "safer" - code integrity
- Can be used as a dictionary key

Group Projects - Initial team planning

Group Projects

https://bcourses.berkeley.edu/courses/1465709/pages/project-overview

Tasks for today

1. Get to know each other
2. Establish a team repository - give ownership to the class organization
3. Create a README file that lists the team members, team name, and initial brainstorming ideas for program the team would like to develop through the project.

End of Meeting #4

For next meeting

- Videos: N/A
- Readings:
    - Lutz Chapter 10: Introducing Python Statements
    - Lutz Chapter 11: Assignment, Expressions, and Prints