Python Types II: Lists, tuples, dicts

Jean Mark Gawron

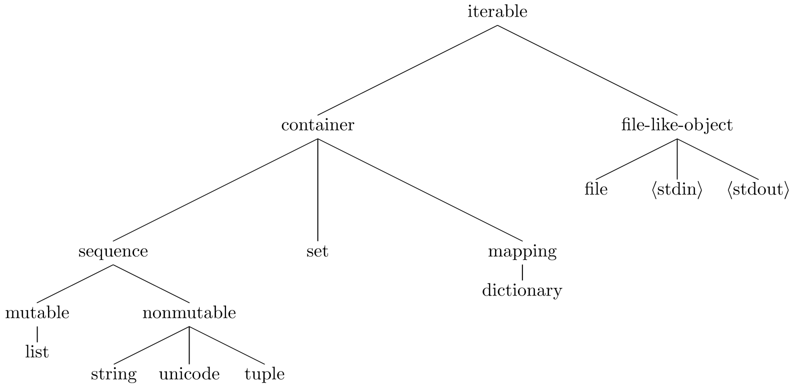
Linguistics 572
San Diego State University

October 7, 2023

Section 1

Containers

Iterables



Python containers fall under **iterables**

Strings revisited

Strings are containers. What do they contain?

```
>>> X = 'abcde'
>>> 'c' in X
True
>>> len(X)
5
```

Every container supports the **in** test. Every container has a computable number of elements (**len** function).

Most important Python Supertypes

- Iterables: Can be looped through
- Containers: Have data (of all types) **in** in them
- Sequences: Data can be retrieved by sequence position

Today we focus on what "containing" data means, on what retrieving by sequence position means, on retrieval in non-sequence containers.

Most important Python Containers

- Sequential
    - List
    - String
    - Tuple
- Non-sequential
    - Dictionary
    - Set

Section 2

List

Creating lists by listing

A list with multiple data types

```
>>> X = [24, 3.14, 'w', [1,'a']]
>>> len(X)                          # 4 items
4
>>> 24 in X                         # container supports in test
True
>>> [1,'a'] in X                    # This list contains a list
True
>>> Y = [100]                       # list with one item
>>> Z = []                          # empty list
```

Creating lists by other means

```
>>> L = list('hi!')
>>> L
['h', 'i', '!']
>>> M = list()
>>> M                                # Empty list again.
[]
```

We often start out by defining an empty list and then writing code to fill it.

Lists as ordered data (sequences)

```
>>> X
[24, 3.14, 'w', [1, 'a']]
>>> X[0]     # 1st element
24
>>> X[1]     # 2nd element
3.14
>>> X[-1]    # last element
[1, 'a']
>>> X[4]     # Raises exception!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

List splices

```
>>> X
[24, 3.14, 'w', [1, 'a']]
>>> X[0:2] # list of 1st and 2nd elements
[24, 3.14]
>>> X[:-1] # list excluding last element
[24, 3.14, 'w']
```

Changing lists I

```
>>> result =  ['January', 'February', 'March', 'April']
>>> result.append('May')
>>> result
['January', 'February', 'March', 'April', 'May']
>>> result =  ['January', 'February', 'March', 'April']
>>> result + ['May']
['January', 'February', 'March', 'April', 'May']
>>> result   # This didnt change result
['January', 'February', 'March', 'April']
>>> new_result = result + ['May']
>>> new_result
['January', 'February', 'March', 'April', 'May']
```

Changing lists II

```
>>> result =  ['January', 'February', 'March', 'April']
>>> result[1] = 'September'  # 2nd element -> 'September'
>>> result
['January', 'September', 'March', 'April']
>>> result[0:2] = ['December', 'February']
>>> result                   # First 2 elements changed
['December', 'February', 'March', 'April']
```

Section 3

Revisiting strings as sequences

Strings are containers

```
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)
34
>>> 'x' in s
True
```

Strings are sequences

```
>>> X = 'dogs'
>>> X[0]     # 1st element
'd'
>>> X[1]     # 2nd element
'o'
>>> X[-1]    # last element
's'
>>> X[4]     # Raises exception!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

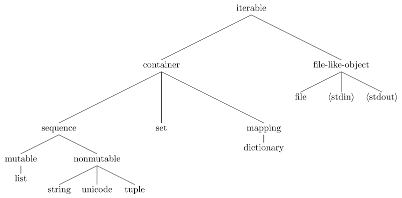
String splices

```
>>> X
'dogs'
>>> X[0:2] # list of 1st and 2nd elements
'do'
>>> X[:-1] # list excluding last element
'dog'
```

Difference 1 between string and list

```
>>> X = 'a2c'
>>> X[1]
'2'
>>> type(X[1]) # Strings contain only strings
<class 'str'>
>>> X[1] + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Difference 2 between string and list



```
>>> X = 'a2c'
>>> X[1] = 'b'     # Strings are immutable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Section 4

Tuples

Tuples are sequences

```
>>> X = (24, 3.14, 'w', 7)    # tuple with 4 items
>>> len(X)
4
>>> 'w'  in X
True
>>> Y = (100,)                # tuple of length 1,  note requir
>>> Z = ()                    # empty tuple
```

Creating tuples by other means

```
>>> L = tuple('hi!')
>>> L
('h', 'i', '!')
>>> M = tuple()
>>> M                                    # Empty list again.
()
```

Tuples as ordered data (sequences)

```
>>> X
(24, 3.14, 'w', 7)
>>> X[0]     # 1st element
24
>>> X[1]     # 2nd element
3.14
>>> X[-1]    # last element
7
>>> X[4]     # Raises exception!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range
```

Tuple splices

```
>>> X
(24, 3.14, 'w', 7)
>>> X[0:2] # list of 1st and 2nd elements
(24, 3.14)
>>> X[:-1] # list excluding last element
(24, 3.14, 'w')
```

Sole diff between tuple & list

Tuples are immutable!

```
>>> result =  ('January', 'February', 'March', 'April')
>>> result[1] = 'September'  # 2nd element -> 'September'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> result
('January', 'February', 'March', 'April')
```

Section 5

Dictionaries

Dictionaries are containers

The dictionary dd has 5 **keys** and 5 **values**. The keys are strings, the values are integers.

```
>>> dd = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> len(dd)
4
>>> 'c' in dd
True
>>> 2 in dd
False
```

Dictionaries are not sequences

```
>>> dd['b']
2
>>> dd[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 1
```

Data is accessed by key, not by a numerical index. In classic python, there is no notion of order in a dictionary.

Keys have unique values

```
>>> dd = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'b':3}
>>> dd
{'a': 1, 'b': 3, 'c': 3, 'd': 4}
```

Note that the key `b` has the last value assigned to it; note that it is perfectly okay for a single value (3) to be associated with multiple keys.

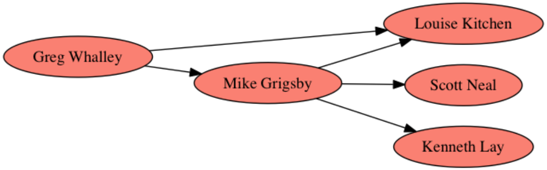
Dictionaries are mappings I

Represent a systematic mapping **from** one kind of data **to** another.

New York	8,175,133
Los Angeles	3, 792,621
Chicago	2,695,598
Houston	2.099,451

```
{'New York':    8175133,
 'Los Angeles': 3792621,
 'Chicago':     2695598,
 'Houston':     2099451
  }
```

Dictionaries are mappings II



Partial email graph for Enron data: A link from person A to person B means person A regularly emails person B

Defining Dictionary of enron emails

```
>>> enron_network = dict()
>>> enron_network['MG']  = ['LK','SN','MG']
>>> enron_network['GW']  = ['LK','MG']
```

Dictionary of enron emails

```
>>> print(enron_network)
{'MG': ['LK', 'SN', 'MG'], 'GW': ['LK', 'MG']}
```

Dictionaries are mutable

```
>>> dd = {'a': 1, 'b': 2, 'c': 3, 'd': 5}
>>> dd
{'a': 1, 'b': 2, 'c': 3, 'd': 5}
>>> dd['d'] = 4
>>> dd
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

Dictionary updates

```
>>> dd = {'a': 1, 'b': 2, 'c': 3, 'd': 5}
>>> ee = {'a': 0, 'b': 2, 'd': 4, 'e':5}
>>> dd.update(ee)
>>> dd
{'a': 0, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

Section 6

Summary of basic container properties

The basic container types

Type	Sequence?	Mutable?	Contents
list	yes	yes	unrestricted
tuple	yes	no	unrestricted
string	yes	no	i in string implies i is a string
dict	no	yes	key must be immutable

Missing above: unordered unrestricted containers:

Completed container types

Type	Sequence?	Mutable?	Contents
list	yes	yes	unrestricted
tuple	yes	no	unrestricted
set	**no**	**yes**	**unrestricted**
frozenset	**no**	**no**	**unrestricted**
string	yes	no	i in string $\Longrightarrow$ i is a string
dict	no	yes	key must be immutable

Some examples to work through

Data type questions

The questions on the next slide are not about persistent storage (saving something on a disk so that it'll still be here after you turn the computer off and on). The questions are about how you should represent the data when it's in the computer's memory and you're preparing to do some processing on it. So the answer is never "in a file". Your answers should be selected from among the types below, but they may be combinations of the types (e.g., a list of dictionaries).

list
tuple
string
dictionary
set

Some examples to work through

1. What data type would you use to represent a phone number?
2. For each member of the class, you need to store their telephone number. What Python data type should you use to represent this? If your answer is a container, say what data type the items in the container are.
3. For each pair of cities in your project, you need to store the traveling distance. What data type should you use? Memory is precious. The distance between Los Angeles and Chicago is the same as the distance between Chicago and Los Angeles.

Some examples to work through (ctd.)

4. An anonymized list of phone numbers to be used for cold calls on election night. Each list contains the phone number of a household likely to vote for our candidate. The list is prioritized so that the households most likely to vote for our candidate come first. Does it change your answer if some of the numbers have extensions?
5. The entire contents of the Constance Garnett translation (into English) of Leo Tolstoy's novel *War and Peace* (566K words, 3.36M characters).
6. The word counts for the Constance Garnett translation (into English) of Leo Tolstoy's novel *War and Peace*.

File IO: The last iterable

Jabberwocky.txt (first 4 lines)

```
Twas brillig, and the slithy toves
      Did gyre and gimble in the wabe:
All mimsy were the borogoves,
     And the mome raths outgrabe.
```

```
>>> text = [ ]
>>> with open("Jabberwocky.txt","r") as fh:
...      for line in fh:
...          text.extend(line.split())
```

File IO (ctd.)

```
>>> text[:6]
[''Twas', 'brillig,', 'and', 'the', 'slithy', 'toves']
```

Making a vocabulary

```
>>> vocab = list(set(text)) # Remove dups, cast into sequence
>>> vocab.sort()            # Put in alphabetical order
>>> vocab[:6]
['All', 'And', 'And,', 'Bandersnatch!”', 'Beware', 'Callay!”']
```