Standard collections library

TEAM INFDEV

Hogeschool Rotterdam
Rotterdam, Netherlands

Introduction

Introduction

Lecture topics

- Building one's collections is often not needed
- Most modern languages couple a standard library with the runtime
- In this lecture we shall list the various built-in collections of Python (≥ 3)

Problem discussion

Problem discussion

Introduction

- How do we represent collections of data in Python?
- What sort of collections are available?
- What properties does each collection offer?

Problem discussion

Introduction

- Collections are not always straightforward **bags** of data
- Collections should be thought of as **bags with structure**

Problem discussion

Bags?

- A bag of data contains multiple values
- The size of the bag is usually variable
- Elements may be accessed via some dynamic mechanism
  - For example, *find the third element*

Problem discussion

Bags **with structure**?

- Structure of data refers to how data is stored
- There are **a lot** of possible structures of data
  - Lack of duplicates
  - Connections between values
  - Order-preservation
  - ...
- We cannot really cover them all, we shall just brush the surface and remind you of the existance of `www.google.com`

Problem discussion

Example structures we shall encounter

Tuple	Fixed number of elements in a fixed order
List	Dynamic number of elements in a fixed order
Sets	Dynamic number of unique elements
Maps	Dynamic number of unique keys mapped to non-unique values

Problem discussion

Example tuples - **(THIS IS NOT CODE)**

- A fixed number of values, kept in a fixed order
- `(0,5)`
- `(1,‘‘Hello!’’)`
- `(‘‘Hello!’’, ‘‘World!’’, 100)`

Problem discussion

Example lists - **(THIS IS NOT CODE)**

- A dynamic number of values, kept in a fixed order
- `[]`
- `[0; 5; 10; 20; 100; 20]`
- `[‘‘Hello!’’; ‘‘World!’’; 100]`

Problem discussion

Example sets - **(THIS IS NOT CODE)**

- A dynamic number of values, without specific order, and no duplicates
- {}
- {0; 5; 10; 20; 100} = {0; 5; 10; 20; 100; 20}
- {``Hello!''; ``World!''; 100}

Problem discussion

Example maps - (THIS IS NOT CODE)

- A dynamic number of keys, each connected to its value[a]
- `[]`
- `[0 ↦ "John"; 1 ↦ "Jack"; ↦ "Jill"]`

[a] The stack and the heap are maps!

General idea

General idea

Introduction

- Python offers a specific data structure for each of these containers
- These data structures are fast to write due to specialized syntax
- These data structures are fast to use due to internal optimizations

General idea

Introduction

- Python offers a specific data structure for each of these containers
- These data structures are fast to write due to specialized syntax
- These data structures are fast to use due to internal optimizations
- These data structures are **not the only option**, sometimes you might need to build your own

General idea

Description

- We will need to learn new syntax and new behaviours
- Keep in mind that this new syntax is not as essential as the basic syntax of Python
- This new batch of syntax is just aesthetics

General idea

Description

- We will need to learn new syntax and new behaviours
- Keep in mind that this new syntax is not as essential as the basic syntax of Python
- This new batch of syntax is just aesthetics
- We could remove it and the language would not lose expressive power

Tuples

Tuples

Making an empty tuple

```
1 empty = ()
```

Tuples

Making a tuple with just one element[1]

```
1 singleton = 'hello',
```

[1] **NOTE the trailing comma!!!**

Tuples

Making a tuple with more than one element

```
1 t = (12345, 54321, 'hello!')
```

Tuples

Breaking it down

```
1 t = (12345, 54321, 'hello!')
2 t_x = t[0]
3 x, y, z = t
```

Tuples

Nesting tuples

```
1 t = (12345, 54321, 'hello!')
2 u = t, (1, 2, 3, 4, 5)
```

Tuples

Pretty printing

```
1 t = (12345, 54321, 'hello!')
2 print(t)
```

Lists

Lists

Making an empty list

```
1 l = []
```

Lists

Making a non-empty list

```
1  a = [-1, 1, 66.25, 333, 333, 1234.5]
```

Finding elements

```
1 a[0]
```

Finding ranges

```
1 a[1:3]
```

Removing elements

```
1 del a[0]
```

Lists

Removing ranges

```
1 del a[1:3]
```

Removing whole range

```
1 del a[:]
```

Lists

Transforming a whole list (`map`)[2]

```
1 squares = list(map(lambda x: x**2, range(10)))
```

[2] Notice the `list` function that finalizes creation of the list.

Lists

List comprehensions **(THIS IS NOT ACTUAL CODE!)**

```
1 [<<element>> for <<i1>> in <<l1>> ... for <<iN>> in <<lN>> ... if <<COND>>]
```

Description

- For each value `i1` in list `l1`
- ...
- For each value `iN` in list `lN`
- If `COND` is true
- Add `element` to resulting list

Lists

Building lists with list comprehensions

```
1 squares = [x**2 for x in range(10)]
```

Lists

Building lists with list comprehensions, multiple iterators, and conditionals

```
1 [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
```

Lists

And a lot of other things...[3]

```
1  list.append(x) #Add an item to the end of the list
2  list.extend(L) #Extend the list by appending all the items in L
3  list.insert(i, x) #Insert an item at a given position
4  list.remove(x) #Remove the first item x from the list
5  list.pop([i]) #Remove the item at the given position in the list
6  list.clear() #Remove all items from the list
7  list.index(x) #Return the index in the list of the first x
8  list.count(x) #Return the number of times x appears in the list
9  list.sort(key=None, reverse=False) #Sort the items of the list in place
10 list.reverse() #Reverse the elements of the list in place
11 list.copy() #Return a shallow copy of the list
```

[3] Just be aware of their existence, learn them as you need!

Sets

Sets

Making an empty set[4]

```
1 p = set()
```

[4] Watch out: **it is not** {}

Sets

Making a non-empty set

```
1 basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
```

Sets

Lookup in a set

```
1 hasOranges = 'orange' in basket
```

Sets

Union and intersection

```
1 inAorInB = a | b
2 inAandInB = a & b
```

Sets

Difference

```
1 inAbutNotB = a - b
2 inAorBButNotBoth = a ^ b
```

Sets

Set comprehensions

```
1 {x for x in 'abracadabra' if x not in 'abc'}
```

Maps

Maps

```
1  tel = {'jack': 4098, 'sape': 4139}
2  print(tel)
3  tel['guido'] = 4127
4  print(tel)
5  print(tel['jack'])
6  del tel['sape']
7  print(tel)
8  tel['irv'] = 4127
9  print(tel)
10 print(list(tel.keys()))
11 print(sorted(tel.keys()))
12 print('guido' in tel)
13 print('jack' not in tel)
14 print(dict([('sape', 4139), ('guido', 4127), ('jack', 4098)]))
15 knights = {'gallahad': 'the pure', 'robin': 'the brave'}
16 for k, v in knights.items():
17   print(k, v)
```

Iterators and generators

Conclusion

Conclusion

Lecture topics

- What problem did we solve today, and how?

This is it!

The best of luck, and thanks for the attention!