

MAT 305: Mathematical Computing

Lecture 5: Collections in Sage

John Perry

University of Southern Mississippi

Fall 2009

Outline

1 Collections in Python

2 Ranges of data

3 Strings

4 Summary

You should be in worksheet mode to repeat the examples.

Collections?

Collection: group of objects identified as single object

- ordered
 - points $(x_0, y_0), (x_0, y_0, z_0)$
 - tuples $(a_0, a_1, a_2, \dots, a_n)$
 - lists $[a_0, a_1, \dots, a_n]$
 - sequences (a_0, a_1, a_2, \dots)
- unordered
 - sets $\{a_0, a_5, a_3, a_2, a_1\}$

Outline

① Collections in Python

② Ranges of data

③ Strings

④ Summary

Python collections

Sage offers several collections standard in Python

- ordered (“sequence types”)
 - tuples, lists
 - access i th element using $[i-1]$
- unordered (“set types”)
 - sets
 - cannot access i th element
 - only one instance of any element

Tuples

Collections in
Python

Ranges of data

Strings

Summary

A **tuple** is an immutable, ordered collection

- *immutable*: cannot change elements
- defined using parentheses

Example

Collections in
Python

Ranges of data

Strings

Summary

```
sage: my_tuple = (1,5,0,5)           4-tuple
sage: my_tuple[2]                     access 3rd element
0
sage: my_tuple[2] = 1                assign to 3rd element?
...Output deleted...
TypeError: 'tuple' object does not support item
assignment
sage: my_tuple
(1,5,0,5)
```

Lists

Collections in
Python

Ranges of data

Strings

Summary

A **list** is a mutable, ordered collection

- *mutable*: can change elements
- defined using square brackets

Example

Collections in
Python

Ranges of data

Strings

Summary

```
sage: my_list = [1,5,0,5]           list of 4 elements
sage: my_list[2]                   access 3rd element
0
sage: my_list[2] = 1              assign to 3rd element?
sage: my_list[2]
1                               no error! access gives new value!
sage: my_list
[1,5,1,5]
```

Sets

Collections in
Python

Ranges of data

Strings

Summary

A **set** is a mutable, unordered collection

- defined using `set` (*tuple or list*)
- define empty set using `set()`
- redundant elements automatically deleted
- elements may not remain in the order you supply them

Example

Collections in
Python

Ranges of data

Strings

Summary

```
sage: my_set = set([1,5,0,5])           set of 4 elements
sage: my_set[2]                         access 3rd element?
...Output deleted...
TypeError: 'set' object is unindexable
sage: my_set                           so what's in there, anyway?
set([0, 1, 5])                        not original list!
```

Does he do any tricks? (1)

Collections in
Python

Ranges of data

Strings

Summary

- sets, tuples, and lists
 - $\text{len}(C)$
number of elements in C
 - $x \in C$
is x an element of C?
- tuples and lists
 - $C.\text{count}(x)$
Number of times x appears in C
 - $C.\text{index}(x)$
First location of x in C
 - $C1 + C2$
join C1 to C2, returned as new tuple/list

Example

Collections in
Python

Ranges of data

Strings

Summary

```
sage: len(my_set)
3
sage: 4 in my_set
False
sage: 5 in my_set
True
sage: my_tuple.count(5)
2
sage: my_list.index(5)
1
sage: my_list + [1,3,5]
[1, 5, 0, 5, 1, 3, 5]
```

How many 5s?

in second location

Does he do any tricks? (2)

Collections in
Python

Ranges of data

Strings

Summary

- lists

- $L.append(x)$
adds x to the end of L
- $L.extend(C)$
appends each element of the collection C
- $L.insert(i, x)$
insert x at $L[i]$, shifting this and subsequent elements back
- $L.pop(i)$
delete $L[i]$ and tell me its value
- $L.remove(x)$
remove first instance of x from L
- $L.reverse()$
reverse the order of elements
- $L.sort()$
*sort the elements of L according to its “natural” order
only a good idea for primitive elements*

Example

Collections in
Python

Ranges of data

Strings

Summary

```
sage: my_list
[1, 5, 0, 5]
sage: my_list.extend((2,4))
sage: my_list
[1, 5, 0, 5, 2, 4]
sage: my_list.insert(3,-1)
sage: my_list
[1, 5, 0, -1, 5, 2, 4]
sage: my_list.pop(3)
-1
sage: my_list.sort()
sage: my_list
[0, 1, 2, 4, 5, 5]
```

A word on inserting

start:

my_list	1	5	0	5	2	4
	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]

```
sage: my_list.insert(3,-1)
```

A word on inserting

start:

my_list	1	5	0	5	2	4
	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]

```
sage: my_list.insert(3,-1)
```

A word on inserting

start:

my_list	1	5	0	5	2	4
	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]

```
sage: my_list.insert(3,-1)
```

my_list	1	5	0	-1	5	2	4
	L[0]	L[1]	L[2]	L[3]	L[4]	L[5]	L[6]

Does he do any tricks? (3)

Collections in
Python

Ranges of data
Strings
Summary

- sets

- $S.add(x)$
adds x to S
- $S.clear()$
removes all elements from S
- $S.difference(C)$, $S.intersection(C)$, $S.union(C)$
set operations: difference $S \setminus C$, intersection $S \cap C$, union $S \cup C$
- $S.isdisjoint(C)$
True iff S and C share no elements
- $S.pop()$
removes and reports random (first?) element of S
- $S.remove(x)$
remove x from S
- $S.symmetric_difference(x)$
set operation: symmetric difference $S \Delta C \cup C \Delta S$

Example

```
sage: my_set = set([1,5,0,5])
sage: my_set.add(4)
sage: my_set
set([0, 1, 4, 5])
sage: my_set.isdisjoint([-1,-2,4])
False
sage: my_set.symmetric_difference([-1,-2,4])
set([-2, -1, 0, 1, 5])
sage: my_set.remove(2)
...Output removed...
KeyError: 2
sage: my_set.remove(1)
sage: my_set
[0, 4, 5]
```

Collections in
Python

Ranges of data

Strings

Summary

Outline

① Collections in Python

② Ranges of data

③ Strings

④ Summary

Tricks with []

Negative indices have meaning:

L[0] L[1] L[2] L[3] L[4]

IndexError

1	5	0	5
---	---	---	---

 IndexError

L[-5] L[-4] L[-3] L[-2] L[-1]

Tricks with []

Negative indices have meaning:

L[0] L[1] L[2] L[3] L[4]

IndexError

1	5	0	5
---	---	---	---

 IndexError

L[-5] L[-4] L[-3] L[-2] L[-1]

Example

```
sage: L = [1,5,0,5]
```

```
sage: L[-1]
```

```
5
```

```
sage: L[-4]
```

```
1
```

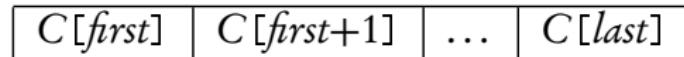
```
sage: L[-5]
```

...Output deleted...

```
IndexError: list index out of range
```

Tricks with [:]

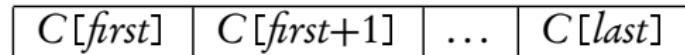
$C[first:last+1]$ specifies subcollection



- omit $first$? \implies start at $C[0]$
- omit $last$? \implies end at $C[-1]$

Tricks with [:]

$C[first:last+1]$ specifies subcollection



- omit $first$? \implies start at $C[0]$
- omit $last$? \implies end at $C[-1]$

Example

sage: L[2:4]	L[2] to L[3]
[0, 5]	
sage: L[:2]	L[0] to L[1]
[1,5]	
sage: L[2:]	L[2] to L[-1]
[0,5]	
sage: L[:]	L[0] to L[-1]
[1,5,0,5]	

The range() command

Collections in
Python

Ranges of data

Strings

Summary

`range(first, last+1)` generates a list with $n = last + 1 - first$ elements where

- $first$ is the first integer in the list
 - default value is 0
- $last$ is the last integer in the list
- $first \geq last$? empty list

Example

```
sage: range(5)
[0, 1, 2, 3, 4]
sage: range(1,5)
[1, 2, 3, 4]
sage: range(3,5)
[3,4]
sage: range(5,5)
[]
sage: range(6,5)
[]
```

Collections in
Python

Ranges of data

Strings

Summary

Outline

① Collections in Python

② Ranges of data

③ Strings

④ Summary

Strings

Collections in
Python

Ranges of data

Strings

Summary

String: ordered collection of characters



- extract elements using []
- join elements using +
- other useful functions on pg. 96 of text

Example

```
sage: name = 'Euler'  
sage: name[2]                                3rd character  
'l'  
sage: name[-1]                               last character  
'r'  
sage: name[0:4]                             first four characters in string  
'Eule'  
sage: name + ' computed'                    add string; notice space  
'Euler computed'
```

The `str()` command

Collections in
Python

Ranges of data

Strings

Summary

`str(x)` where

- *x* is any object that can be turned into a string
- Sage will turn a *lot* of objects into strings!

Example

Numbers:

```
sage: name + ' computed' + ' e**(i*pi) + 1 = '
      + str(0)
'Euler computed e**(i*pi) + 1 = 0'
```

Example

Collections in
Python

Ranges of data

Strings

Summary

Numbers:

```
sage: name + ' computed' + ' e**(i*pi) + 1 = '
      + str(0)
'Euler computed e**(i*pi) + 1 = 0'
```

Equations: (after “obvious” simplifications!)

```
sage: name + ' computed ' + str(e**(i*pi) + 1 == 0)
'Euler computed 0 == 0'
```

Collections in
Python

Ranges of data

Strings

Summary

Outline

① Collections in Python

② Ranges of data

③ Strings

④ Summary

Summary

- Through Python, Sage offers several kinds of collections
 - tuples, lists, sets
- Operations
 - [] for extraction
 - negatives allowed
 - [:] gives subcollections
 - usual mathematical operations on sets
 - others supplied by Python
- Strings allow lists of characters
 - `str(x)` produces “obvious” string representation of x