

Outline

❑ Built-in Data Structures

- List
- Tuple
- Set
- Dictionary

❑ Control Flow

- If ... else, while, for

❑ Iterations

Introduction:

❑ Sequence Types

- Sequences are *containers* that hold objects
 - Built-in Python Data Structures
 - Finite, ordered, indexed by integers
- Tuple
 - An *immutable* ordered sequence of items
 - Items can be of mixed types, including collection types
- Strings
 - An *immutable* ordered sequence of chars
 - Conceptually very much like a tuple
- List
 - A *Mutable* ordered sequence of items of mixed types
- Dictionary
 - A *Mutable* ordered sequence of items with two elements (key and data) of mixed types

❑ Set

- A *Mutable* unordered sequence of items of mixed types without duplicated items

Lists

Syntax: `[elem1, elem2, ...]`

- Ordered sequence of any type (mixed types ok)
- Mutable

```
>>> list1 = [1, 'hello', 4+2j, 123.12]
```

```
>>> list1
```

```
[1, 'hello', (4+2j), 123.12]
```

```
>>> list1[0] = 'a'
```

```
>>> list1
```

```
['a', 'hello', (4+2j), 123.12]
```

Repeating

Concatenation: `list1 + list2`

```
>>> [1, 'a', 'b'] + [3, 4, 5]  
[1, 'a', 'b', 3, 4, 5]
```

Repetition: `list * count`

```
>>> [23, 'x'] * 4  
[23, 'x', 23, 'x', 23, 'x', 23, 'x']
```

Elements

```
>>> list = [ "apple", "banana" ]
```

Append item to end

```
>>> list.append( "durian" )
```

Append another list

```
>>> list.extend( list2 )
```

- *Same as* `list + list2`

Insert item anywhere

```
>>> list.insert( 0, "artichoke" )
```

```
>>> list.insert( 2, "carrot" )
```

Like Java `list.add(n, item)`

Not so object-oriented `len()`

`len()` returns length of a list

```
>>> list = [ "a", "b", "c" ]
```

```
>>> len( list )
```

3

Removing Elements from a List

```
>>> list = [ "a" "b", "c", "b" ]
```

- Remove a matching element (w/o returning it)

```
>>> list.remove( "b" )
```

Throws exception if argument is not in the list

- Remove last element and return it

```
>>> list.pop( )
```

```
'b'
```

- Remove by position

```
>>> list.pop( 1 )    # 'b' removed already  
'c'
```

Indexing

Syntax: `list[n]`

- Positive indices count from the left: `list[0]`
- Negative indices count from the right: `list[-1]`

| | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| <i>a</i> | <i>b</i> | <i>c</i> | <i>d</i> | <i>e</i> | <i>f</i> | <i>g</i> |
| -7 | -6 | -5 | -4 | -3 | -2 | -1 |

| | |
|---------------------------|--------------------------------|
| <code>list[0] == a</code> | <code>sequence[-1] == g</code> |
| <code>list[2] == c</code> | <code>sequence[-2] == f</code> |
| <code>list[6] == g</code> | <code>sequence[-7] == a</code> |

List Slicing: get a sublist

`list[m:n]` return elements `m` up to `n` (exclusive)

syntax for both `strings` and `lists`

```
>>> x = [0, 1, 2, 3, 4, 5, 6, 7]
>>> x[1:4]
[1, 2, 3]
>>> x[2:-1]
[2, 3, 4, 5, 6]
# Missing Index means start or end of list
>>> x[:2]
[0, 1]
>>> "Hello nerd"[3:]
lo nerd
```

Remove element from a list

`list.remove(element)`

Removes the first occurrence of `element` in `list`

Example:

```
>>> list2 = [0, 1, 3, 4, 3, 5]
```

```
>>> list2.remove(3)
```

```
>>> list2
```

```
[0, 1, 4, 3, 5]
```

`del list[index]`

Delete element by index.

Remove & return an element

Syntax: `list.pop([index])`

- Remove and return item at position `index` from `list`
- Without an argument: remove last item

Example:

```
>>> list1 = [1, 'hi', 2, 9, 'the end']
>>> list1.pop(1)
hi
>>> list1.pop()
the end
>>> list1
[1, 2, 9]
```

Sorting a list

`List.sort([comparator])`

Sort `List` *in place*. Result is applied to the list!

Example:

```
>>> list3 = [4, 12, 3, 9]
```

```
>>> list3.sort()
```

```
>>> list3
```

```
[3, 4, 9, 12]
```

Count or find elements in a list

```
list.count( element )
```

count number of occurrences of element.

```
n = list.index( element )
```

return index of first occurrence of element.

Throws **ValueError** if element is not in list.

Common list methods

Methods with Description

[list.append\(obj\)](#)

Appends object obj to list

[list.count\(obj\)](#)

Returns count of how many times obj occurs in list

[list.extend\(seq\)](#)

Appends the contents of seq to list

[list.index\(obj\)](#)

Returns the lowest index in list that obj appears

[list.insert\(index, obj\)](#)

Inserts object obj into list at offset index

[list.pop\(obj=list\[-1\]\)](#)

Removes and returns last object or obj from list

[list.remove\(obj\)](#)

Removes object obj from list

[list.reverse\(\)](#)

Reverses objects of list in place

[list.sort\(\[func\]\)](#)

Sorts objects of list, use compare func if given

Tuples

Immutable list

Syntax: (elem1, elem2, ...)

A tuple cannot be changed.

Example:

```
>>> tuple1 = (1, 5, 10)
```

```
>>> tuple1[2] = 2
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#136>", line 1, in ?
```

```
    tuple1[2] = 2
```

```
TypeError: object doesn't support item  
assignment
```

Converting between list and tuple

```
>>> list1 = ['a', 'b', 'c']
```

```
>>> tuple1 = tuple( list1 )
```

```
>>> type( tuple1 )
```

```
<class 'tuple'>
```

```
>>> tuple2 = ('cat', 'dog')
```

```
>>> list2 = list(tuple2)
```

len - Length of an object

`n = len(object)`

len is not object-oriented.

Return the length of object

Examples

```
>>> list1 = [1, 2, 3, 4, 5]
```

```
>>> len(list1)
```

5

```
>>> string1 = "length of a string"
```

```
>>> len(string1)
```

18

Multiple assignment using tuples

```
(a,b,c) = (10, 20, 50)
```

```
>>> b
```

```
20
```

This can be used in **for** loops.

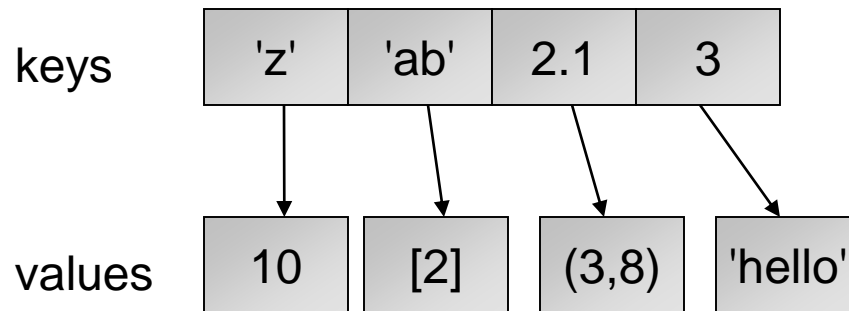
```
points = [ [1,0,3], [0.2,0.9,0.4], [1,2,7] ]  
for [x,y,z] in points:  
    r = math.hypot(x,y)  
    print("radius of (%f,%f) is %f" % (x,y,r) )
```

Dictionary: mapping key to value

A **mapping** of keys to values

Associate a key with a value

Each key must be unique



Using Dictionaries

Syntax: `dict = {key1: value1, key2: value2, ...}`

```
>>> dict = {'a': 1, 'b': 2}
```

```
>>> dict
```

```
{'a': 1, 'b': 2}
```

```
>>> dict['a']
```

```
1
```

```
>>> dict['b']
```

```
2
```

```
>>> dict[3] = 'three'
```

```
>>> dict
```

```
{'a': 1, 'b': 2, 3: 'three'}
```

Set

An unordered collection, without duplicates (like Java).

Syntax is like dictionary, but no ":" between key-value.

```
>>> aset = { 'a', 'b', 'c' }
>>> aset
{'a', 'c', 'b'}
>>> aset.add('c')      # no effect, 'c' already in set
>>> aset
{'a', 'c', 'b'}
```

Set Methods

| | |
|---|-------------------------------------|
| <code>set.discard('cat')</code> | remove cat. No error if not in set. |
| <code>set.remove('cat')</code> | remove cat. Error if not in set. |
| <code>set3 = set1.union(set2)</code> | doesn't change set1. |
| <code>set4 = set1.intersection(set2)</code> | doesn't change set1. |
| <code>set2.issubset(set1)</code> | |
| <code>set2.issuperset(set1)</code> | |
| <code>set1.difference(set2)</code> | element in set1 not set2 |
| <code>set1.symmetric_difference(set2)</code> | xor |
| <code>set1.clear()</code> | remove everything |

Test for element in Set

item in set

```
>>> aset = { 'a', 'b', 'c' }
```

```
>>> 'a' in aset
```

```
True
```

```
>>> 'A' in aset
```

```
False
```

Flow Control

```
if condition :  
    body  
elif condition :  
    body  
else:  
    body
```

```
if x%2 == 0:  
    y = y + x  
else:  
    y = y - x
```

```
while condition:  
    body
```

```
while count < 10:  
    count = 2*count
```

```
for name in iterable:  
    body
```

```
for x in [1,2,3]:  
    sum = sum + x
```

range: create a sequence

```
range([start,] stop[, step])
```

Generate a list of numbers from `start` to `stop`
stepping every `step`

`start` defaults to 0, `step` defaults to 1

Example

```
>>> range(5)
```

```
[0, 1, 2, 3, 4]
```

```
>>> range(1, 9)
```

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

```
>>> range(2, 20, 5)
```

```
[2, 7, 12, 17]
```

for loop using range()

Use `range` to generate values to use in for loop

```
>>> for i in range(1,4):  
        print(i)
```

1

2

3

loop iteration using `continue`

skip to next iteration of a loop

```
for x in range(10):  
    if x%2 == 0:  
        continue  
    print(x)
```

```
1  
3  
5  
7
```

break

break out of the inner-most loop

```
for number in range(10):  
    if number == 4:  
        print ('Breaking')  
        break  
    else:  
        print (number)
```

0

1

2

3

Breaking

Iteration

Most data structures can be iterated over in the same way:

```
mylist = ['a', 'b', 'c']  
for item in mylist:  
    print(item)
```

```
mytuple = ('a', 'b', 'c')  
for item in mytuple:  
    print(item)
```

```
dict = {'a': 10, 'b': 15}  
for key in dict:  
    print(key, dict[key])
```

Note we don't need the index of the element to access it.

When iterating over a dictionary like this, we iterate over the keys.

Iteration

We can also iterate over indices:

```
for i in range(4):  
    print (i)
```

0 1 2 3

```
for i in range(1, 10, 2):  
    print (i)
```

1 3 5 7 9

```
mylist = ['a', 'b', 'c']  
for i in range(len(mylist)):  
    print (i, mylist[i])
```

0 'a'
1 'b'
2 'c'

```
mylist = ['a', 'b', 'c']  
for idx, item in enumerate(mylist):  
    print (idx, item)
```

0 'a'
1 'b'
2 'c'

List Comprehensions

Input: `nums = [1, 2, 3, 4, 5]`

Goal: `sq_nums = [1, 4, 9, 16, 25]`

Here's how we could already do this:

```
sq_nums = []
```

```
for n in nums:
```

```
    sq_nums.append(n**2)
```

Or... we could use a comprehension:

square brackets show
we're making a list

```
sq_nums = [n ** 2 for n in nums]
```

apply some operation
to the loop variable

loop over the specified
iterable

More List Comprehensions


Template:

```
new_list = [f(x) for x in iterable]
```

```
words = ['hello', 'this', 'is', 'python']
```

```
caps = [word.upper() for word in words]
```

```
powers = [(x**2, x**3, x**4) for x in range(10)]
```



Remember this doesn't have to be a list! Can be any iterable.

Summary

❑ Built-in Data Structures

- List
- Tuple
- Set
- String
- Dictionary

❑ Control Flow

- If ... else, while, for

❑ Iteration