Basic Python Programming

Yilin Zheng

Dept. of Computer Science and Engineering

July 3, 2017

Outline I

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Outline II

- Set
- Reference
- String

Modules

- A **.py** file, stores functions, variables, and objects
- Use `import` to import modules before use

Byte-complied .pyc

- This kind of files ends with **.pyc** and can accrelerate the process

from...import

- Use `from Module name import sth.` to import parts of variables or classes or functions within the module
- We **don't need to type the module name** every time referring to it
- Use `*` to represent all in the module

__name__

- Use `__name__` to refer to the module's name

Create own modules

- Create modules by **using Python script**
- Use `import` or `from...import` to use it

dir

- **Without arguments**, it can return **a list of names in current local scope**
- **With an arguments**, it returns **a list of valid attributes for that object**

Data structure

- Data structure is **a particular way of organizing data** so that it can be used efficiently
- There are four built-in data structures in python: **list**, **tuple**, **dictionary**, **Set**

Class and Object

- Here some data structures use objects, you can just regard it as creating an instance of the object, so **don't care too much about them temporarily**
- Classes are **classifications of different objects**, since classes have methods as well, we can operate on the instance
- More detail will be talked later slides

List

- **List** is an example which use the object
- It is an array of items enclosed by **square brackets** [ ]
- The items can be **numbers, characters, strings and so on**, but **all the items should be the same type** and **seperated** by **comma** ,
- We can refer to the items in a list by using **index beginning from zero(0)**
- Example 1: a list of integers

```
>>> list1 = [1,2,3,4]
>>> list1[0]
1
>>> list1[3]
4
>>> list1[4] # this will go wrong
```

List

- Example 2: a list of strings

```
>>> list2 = ['a', 'b', 'c', 'd']
>>> list2[2]
'c'
>>>list3 = ['Python', 'is', 'so', 'easy']
>>> print(list3[0])
Python
```

Operations on list

- Some operation can be performed on list: functions like `len`, sort, `sorted`, `del`, `append`; using `for` to iterate the items

```
>>> len(list1) # return the number of items,
                                also the length of
                                the list
4
>>> list3.sort() #for strings, sort them
                                alphabetically(
                                captial letter is
                                ahead of lowercase)
>>> list3
['Python', 'easy', 'is', 'so']
```

Operations on list

```
>>> list1=[4,7,1,5,8,0]
>>> list1.sort() #for number list, sort in
                                ascending order
>>> list1 #original list changed
[0, 1, 4, 5, 7, 8]
>>> list1.append(10) #add 10 at the tail
>>> list1
[0, 1, 4, 5, 7, 8, 10]
>>> del(list1[6]) # delete 10 from the list
>>> list1
[0, 1, 4, 5, 7, 8]
>>> list2 = [23, 45, 12, 4, 10, 100]
>>> sorted(list2) #it will return a sorted
                                list
[4, 10, 12, 23, 45, 100]
>>> list2 #original list has no change
[23, 45, 12, 4, 10, 100]
```

Operations on list

- These two blocks return the same outcome

```
#!/usr/bin/python # Filename: iterate_list.py
list1 = ['Fish', 'or', 'cut', 'bait']
for i in list1:
    print(i)
```

```
#!/usr/bin/python # Filename: iterate_list.py
list1 = ['fish', 'or', 'cut', 'bait']
for i in range(0, 4):
    print(list1[i])
```

Tuple

- **Tuple** is somthing like list but **the items in it cannot be changed**(the same as string)
- Use **parenthesis** ()to enclose items instead of square brackets
- When we want the items to be used without changes, tuple is a good choice

```
>>> tuple1 = (1,2,3,4,5)
>>> tuple1[0] = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support
                                item assignment
```

Operations on Tuple

- functions like `len`, `sorted`
- Iteration on the items

```
>>> tuple1 = (1,2,3,4,5)
>>> len(tuple1)
5
>>> sorted(tuple1) #return a list consists of
                   the tuple
[1, 2, 4, 5, 8] #this is a list rather than a
                 tuple
```

```
#!/usr/bin/python # Filename: iterate_tuple.py
tuple1 = (1, 7, 2, 9, 6)
for i in tuple1:
    print(i)
```

Dictionary

- **Dictionary** is much different from list or tuple, whose items are **enclosed by braces** {
- Items are still **seperated by comma**, and every item consists of two parts: **key** and **value**, which are seperated by **colon** :
- **Key** should be **unique** and **cannot be changed**, while **value** has **no these limitations**
- **Use key to get value**

```
>>> dict1 = {'Name':'Kangkang', 'Age':18, '
                                Interests':'Video
                                Games'}
>>> dict1
{'Name': 'Kangkang', 'Age': 18, 'Interests': '
                                Video Games'}
>>> dict1['Name'] #use key to get value
'Kangkang'
```

Operations on dictionary

- functions: len, sorted, del
- Iterations on items

```
>>> del(dict1['Interests']) #del the item
                            through key
>>> dict1
{'Name': 'Kangkang', 'Age': 18}
>>> sorted(dict1) #sort keys alphabetically
['Age', 'Interests', 'Name'] #return a tuple

#!/usr/bin/python
# Filename: iterate_dict.py
dict1 = {'Name':'Kangkang', 'Age':18, '
                            Interests':'Video
                            Games'}
for i in dict1:
    print(dict1[i])
```

Sequence

- **List**, **tuple** and **dictionary** are sequence
- Two features about sequence: **Index operator** and **Slice operator**
- Use index operator to **find a specific item**
- Use slice operator to **get a part of sequence**
- Index operator: `[index]`
- Slice operator: `[start:end]` or `[start:end:step]`

```
>>> tuple1=[1,3,5,7,9]
>>> tuple1[1:3]
[3, 5]
>>> tuple1[-1:] #last item
[9]
>>> tuple1[:-2] #all except last two items
[1, 3, 5]
```

Slice operator

- Detail about the slice

```
seq[start:end] # items start through end-1(
                           index)
seq[start:]    # items start through the rest
seq[:end]      # items from the beginning
                           through end-1
seq[:]         # a copy of the whole sequence
seq[start:end:step] #start~end-1 by step, by
                           default is 1
```

- if step >0, from left to right; if step < 0, from right to left (**cannot be zero**)

```
>>> tuple1[4:1:-1]
[9, 7, 5]
```

Slice operator

- if negative, it will take out the slice in the opposite

```
>>> tuple1
[1, 3, 5, 7, 9]
>>> tuple1[:-2]
[1, 3, 5]
>>> tuple1[-2:]
[7, 9]
```

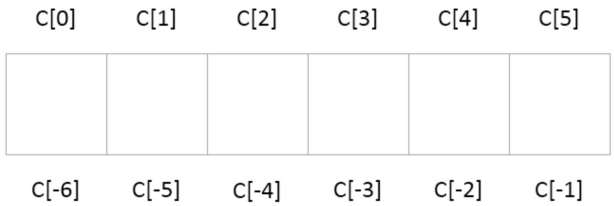

Figure: sequence

Set

- Sets are **unordered collections** of simple objects
- We ues set when **the existence** of an object in a collection is **more important** than **the order** or **how many times it occurs**
- We can use sets to **test for membership**, **find the intersection between two sets**, and so on.
- Use `set` to create sets

```
>>> fruit = set(['watermelon', 'banana'])
>>> fruit
{'banana', 'watermelon'} #alphabetic order
>>> fruit = set({'watermelon', 'banana'}) #
                                same result
>>> fruit = set(('watermelon', 'banana')) #
                                same result
>>> fruit = set('watermelon', 'banana') #error
```

Operations on set

- functions and methods: `len`, add, remove, `copy`
- Operator: `in`

```
>>> fruit.add('apple')
>>> fruit
{'banana', 'apple', 'watermelon'}
>>> fruit.remove('banana')
>>> fruit
{'apple', 'watermelon'}
>>> fruit_copy = fruit.copy()
>>> fruit_copy
{'apple', 'watermelon'}
>>> 'apple' in fruit
True
>>> 'durio zibethinus' in fruit
False
```

Reference

- When you create an object and assign a varible to it, the variable only **refers to** the object and doesn't represent the object itself
- The variable name points to the part of memory the object stored, which is called **binding the name to the object**

```
#!/usr/bin/python # Filename: reference.py
fruit = ['apple', 'mango']
print("fruit is", fruit)
fruit_ref = fruit # just another name
print("fruit_ref is", fruit_ref)
print("Change fruit by adding 'watermelon'")
fruit.append('watermelon')
print("new fruit is", fruit)
fruit_ref = fruit
print("now fruit_ref is", fruit_ref)
```

More on string

- All the strings is the objects of class **str**
- Some methods of strings: `len`, `startswith`, `in`, `find`, `join`
- More on ▸ String Methods

```
>>> str1 = 'dansriaveruavervaer'
>>> str1.startswith('dan')
True
>>> 'a' in str1
True
>>> str1.find('er')
8
>>> delimiter = '_'
>>> delimiter.join(str1)
'd_a_n_s_r_i_a_v_e_r_u_a_v_e_r_v_a_e_r'
```

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY