CSCI2040:
Lists and Tuples

Yim-Pan CHUI

Computer Science & Engineering Department

The Chinese University of Hong Kong

Strings

- Text is most common form of data in our daily life
- In Python, they are stored as a sequence of characters internally
- The length of a string can be varied during program execution
- Use single quote 'or double quote " to delimit

Strings

- Enclosed either by single quote 'or double quote "

- >>> 'doesn\'t'
  "doesn't"
  >>> "doesn't"
  "doesn't"
  >>> '"Yes," \nhe said.'
  '"Yes," \nhe said.'
  >>> print ('"Yes," \nhe said.')
  "Yes,"
  he said.
  >>> "\"Yes,\" he said."
  '"Yes," he said.'
  >>> '"Isn\'t," she said.'
  '"Isn\'t," she said.'

Use \ to tell not delimit character, \ is called escape char

\n move the cursor to next line, a.k.a newline character

Another way to print " when delimit is also "

Escape character	Prints as
\'	Single quote
\"	Double quote
\t	Tab
\n	Newline (line break)
\\	Backslash

Strings

- Concatenated or repeated with + and * respectively

```
>>> word = 'CUHK'+ 'super'
>>> word
'CUHKsuper'
>>> '['+word*3 + ']'
'[CUHKsuperCUHKsuperCUHKsuper]'
```

Strings

- Access can be in subscripted notation
  ```
  >>> word[2]
  'H'
  ```

- substrings can be specified with the slice notation
  ```
  >>> word[0:2]
  'CU'
  >>> word[:2]
  'CU'
  >>> word[1:]
  'UHKsuper'
  ```

- For non-negative indices, the length of a slice is the difference of the indices, if both are within bounds

Strings

- Immutable – can't be modified once created

```
>>> word[0] = 2
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    word[0] = 2
TypeError: 'str'object does not support item assignment
```

- But we can create a new string in this case

```
>>> word[0:3]+ " course"
'CUH course'
```

Splitting Strings

- useful to break a string into, say words
- takes one argument, the character that separates the parts of the string

```
1 message = "CSCI2040 is really boring !!!, really very boring."
2 words = message.split(' ')  # split uses ' '(or space) as separator
3 print(words)
4
5 words = message.split('really')  # split uses 'really'as separator
6 print(words)
7
8
```

```
['CSCI2040', 'is', 'really', 'boring', '!!!,', 'really', 'very', 'boring.']
['CSCI2040 is ', ' boring !!!, ', ' very boring.']
```

String functions

Note the use of '.'to call the string function

- Operation Description

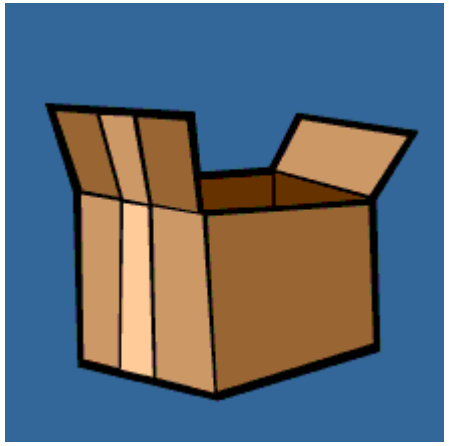
s.capitalize()	capitalize the first character of s
s.count(sub)	count number of occurrence of sub in s
s.find(sub)	find first index of sub in s, or -1 if not found
s.index(sub)	find first index of sub in s, or raise a ValueError if not found
s.rfind(sub)	find last index of sub in s, or -1 if not found
s.rindex(sub)	find last index of sub in s, or raise a ValueError if not found
s.lower()	convert s to lowercase
s.split()	return a list of words in s
s.join(lst)	join a list of words into a single string with s as separator
s.strip()	strips leading/trailing white space from s
s.upper()	convert s to upper case
s.replace(old,new)	replace all instances of old with new in string

What is a list?

- A list is a collection of things!

Ordinary Variable

Like a box for storing one value



List

Like a cabinet containing many drawers.

Each drawer stores one value.

We can refer to each drawer as 1st drawer, 2nd drawer, 3rd drawer, etc.

Lists

- A compound data types that group a number of comma separated values
  >>> squares = [1, 4, 9, 16, 25]

- Access to individual entry is possible through index
  >>> squares[1]
  4

- Note that index of list start from 0

Lists

- Data type of list entries can be of different
```
>>> squares[1] = 'Hello'
>>> squares
[1, 'Hello', 9, 16, 25]
```

- Can index backward, -1 means the last item
```
>>> squares[-1]
25

>>> squares[-4] = 4
>>> squares
[1, 4, 9, 16, 25]
```

Lists

- Also support slicing operation
- >>> squares[-3:]
  [9, 16, 25]

- >>> squares[:3]
  [1, 4, 9]
- >>> squares[:]
  [1, 4, 9, 16, 25]

Lists

- Support concatenation
  >>> squares = squares + [36, 49, 64, 81, 100]
  >>> squares
  [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

- Can also add entry through built-in methods
  >>> squares.append(121)
  >>> squares[10]
  121

Lists

- Changes to slice is also possible
  >>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

  >>> letters[2:5] = ['C', 'D', 'E']
  >>> letters
  ['a', 'b', 'C', 'D', 'E', 'f', 'g']

- Removing slice
  >>> letters[2:5]=[]
  >>> letters
  ['a', 'b', 'f', 'g']

Lists

- Sometimes we may want to know how many entries are there in a list
- Built-in function len() can help

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> len(letters)
7
```

for statement

- Iterate over items in a sequence i.e. lists
- Best suit operating on list

```
1 a = ['HKU', 'CUHK', 'UST']
2 for x in a:
3     print (x, len(x))
4
5
6
7
8
9
```

```
HKU 3
CUHK 4
UST 3
```

for statement

- range() function to help iterate over number sequence similar to while
- Note it start from 0

```
1 for i in range(5):
2     print (i)
3
4
5
6
7
8
9
```

```
Output
0
1
2
3
4
```

for statement

- range() function combine with len() can iterate over elements in a list
- We retrieve the item through their index here

```
1 a = ['Mary', 'had', 'a', 'little', 'lamb']
2 for i in (range(len(a))):
3     print (i, a[i])
4
5
6
7
8
9
```

```
Output
0 Mary
1 had
2 a
3 little
4 lamb
```

More about range()

- You can have more precise control over range()
- range(initial, final, step)

- range(5, 10)
  5, 6, 7, 8, 9

- range(0, 10, 3)
  0, 3, 6, 9

- range(-10, -100, -30)
  -10, -40, -70

Lists

- *we can't use a number larger (or less than) the dimension of the list*

```
cse_teachers = ['John C.S. Lui', 'Patrick P.C. Lee', 'James Cheng', 'Wei Meng']
print ("first element in the list is =", cse_teachers[6])
#print ("first element in the list is =", cse_teachers[-7])
```

```
Traceback (most recent call last):
  File "/COURSE/2040/list1.py", line 5, in <module>
    print ("first element in the list is =", cse_teachers[6])
IndexError: list index out of range
```

Enumerating a list

```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'james cheng', 'wei Meng']
print ("------- use enumerate -----------")
for index, teacher in enumerate(cse_teachers):
    position = str(index)
    print("Position: " + position + " Teacher: " + teacher.title())
print ("------- use list.index(value) -----------")
for teacher in cse_teachers:
    print("Position:", cse_teachers.index(teacher), "Teacher: " + teacher)
```

```
------- use enumerate -----------
Position: 0 Teacher: John C. S. Lui
Position: 1 Teacher: Patrick P. C. Lee
Position: 2 Teacher: James Cheng
Position: 3 Teacher: Wei Meng
------- use list.index(value) -----------
Position: 0 Teacher: john c. s. lui
Position: 1 Teacher: patrick p. c. lee
Position: 2 Teacher: james cheng
Position: 3 Teacher: wei Meng
```

Testing for item membership

```
cse_teachers = ['john c. s. lui', 'patrick p. c. lee', 'y.p.chui', 's.h.or']

print ("Is 'john c. s. lui'in the list?", 'john c. s. lui'in cse_teachers)
print ("Is 'John C. S. Lui'in the list?", 'John C. S. Lui'in cse_teachers)
```

```
Is 'john c. s. lui'in the list? True
Is 'John C. S. Lui'in the list? False
```

List Operations

Operation	Description
s.append(x)	appends element x to s
s.extend(ls)	appends list s with ls
s.count(x)	count number of occurrence of x in s
s.index(x)	return index of first occurrence of x
s.pop()	return and remove last element from s
s.pop(i)	return and remove element i from s
s.remove(x)	search for x and remove it from s
s.reverse()	reverse element of s in place
s.sort()	sort elements of s into ascending order
s.insert(i, x)	inserts x at location i

Tuples

- Similar to list, but form using parenthesis
- Immutable, major difference compare with list
- Non-mutable operations in list can apply to tuple

```
>>> tup = (1,7,3,1,7,3)
>>> 3 in tup
True
>>> list(tup)
[1, 7, 3, 1, 7, 3]
>>> tuple(['a', 'b', 'c'])
('a', 'b', 'c')
```

Tuples

- Comma operator implicitly creates a tuple

```
>>> 'a','b','c'
('a', 'b', 'c')
```

- Application in function returning more than 1 result

```
def minAndMax( info ):
    return (min(info), max(info))
>>> x, y = minAndMax( 'abcd')
>>> x
'a'
>>> y
'd'
```

What good list & strings brings?

- Consider the following problem:
  Write a program to check whether an input string is a strong password, which has to contain…

1. lower case letters,
2. upper case letters,
3. digits,
4. special characters, and
5. the length has to be no shorter than 12.

- We would do this by testing one by one

What good list & strings brings?

- In C language, we would proceed to test the ASCII code of input string character by character

- ```
  char pw[100];
  int cnt=0;

  scanf("%s", pw);
  for (i=0; i<strlen(pw); i++)
    if (islower(pw[i])  cnt++;
  if (cnt<=0) printf("No lower case ");
  ```

Need to think of character, loop , character function
Many low level thinking
  ```

What good list & strings brings?

- In Python, the thinking is different

```
password=input("Enter your password: ")
has_lowercase = False
lowercase='abcdefghijklmnopqrstuvwxyz'
for c in password:
        has_lowercase = has_lowercase or (c in lowercase)
if not has_lowercase:
        print('No lowercase ')
```