ECE326

PROGRAMMING LANGUAGES

Lecture 3 : Python Sequence Types

Kuei (Jack) Sun
ECE
University of Toronto
Fall 2020

Sequence

- An ordered collection of values
- Python list, string, tuple, range, …etc
- Repetition of elements is allowed
  - E.g., in a string, letter *a* can appear more than once
- Provides mapping from index to value
  - Like in C, uses zero-based index

Python String

- Similar to C++ std::string
- Can be created with single or double quote

```
>> a = ‘hello “world”’          # no need to escape
>> print(a)
hello “world”
>> print(“good \”bye\””)        # need to escape
good “bye”
```

- Strings are *immutable*
  - Cannot be changed once assigned
  - Copy is made for every operation

String Method

- strip()
  - Remove whitespace from both sides
- startswith() / endswith()
  - Checks if string starts/ends with substring
- replace()
  - Replace *all* instances of a substring in string
- C-style (printf) format string

```
>> "hello %s #%d"%('world', 42)
'hello world #42'
```

Python List

- Similar to C++ std::vector – more powerful
  - Can place objects of different types within

```
>> a = [ 1, 2.5, “hello” ]  # common initialization
>> list()                   # another way (empty)
[]
```

- Lists are *mutable*, they can be updated

```
>> a.pop()       # remove last element and return it
“hello”
>> a
[1, 2.5]
>> a.append(3)   # add element to end of list
>> a
[1, 2.5, 3]
```

List Methods

- Insertion

```
>> a = [2, 4, 6, 0, 1]
>> a.insert(0, 6)      # insert 6 to index 0
>> a
[6, 2, 4, 6, 0, 1]
```

- Remove by index

```
>> del a[1]            # del expr is a statement
>> a                   # a.pop(1) is an expression
[6, 4, 6, 0, 1]
```

- Remove by value

```
>> a.remove(0)         # removes first occurrence of 0
>> a
[6, 4, 6, 1]
```

Alias

- Different names referring to same memory location
  - Problem: update one implicitly changes the other
  - Sometimes unintentional (frequent source of bugs)

```
>> a = b = []
>> a
[]
>> b
[]
>> a.append(5)
>> a
[5]
>> b   # why?
[5]
>> a = [1, 2, 3]
```

```
>> b = a    # assignment by reference
>> b[1] = 4 # update element
>> a
[1, 4, 3]
>> import copy
>> d = copy.copy(a)
>> d[0] = 5
>> a
[1, 4, 3]
>> d
[5, 4, 3]
```

Solution: make a copy of *a*

String and List Methods

- Tokenize

```
>> “hello big world”.split(‘ ’)
[‘hello’, ‘big’, ‘world’]
```

- Join a list of string using a delimiter

```
>> ‘-’.join([‘hello’, ‘big’, ‘world’])
‘hello-big-world’
```

- Merge with another list

```
>> a = [5, 9]
>> a.extend([1, 2])
>> a
[5, 9, 1, 2]
```

- Sort list

```
>> a = [5, 9, 1, 2]
>> a.sort()
>> a
[1, 2, 5, 9]
```

Tuple

- Same as list, except *immutable*

```
>> a = 1, 2, “hello”, 4
>> a
(1, 2, “hello”, 4)
>> a[1] = 7
TypeError: 'tuple' object does not support item assignment
```

- Can do neat tricks
  - Swap

```
>> a = 3
>> b = 6
>> a, b = b, a
>> a, b
(6, 3)
```

  - Packing/Unpacking

```
>> foo()
(5, 7)
>> x, y = foo()
>> x
5
```

```
def foo():
  return 5, 7
```

Common Operations

On Sequence Types

Index Operator

- Returns *n*th element of the sequence
  - syntax: *sequence*[n]

```
>> b = [2, 3, 5, 7, 11, 13, 17]
>> b[2]
5
>> b[7]
IndexError: list index out of range
>> b[-1]  # returns last element
17
>> b[-8]
IndexError: list index out of range
```

- For List (mutable), can update element

```
>> b[-1] += 1
```

Slicing

- Extracts subset of elements from sequence
  - *sequence*[i:j:k], *i*: start, *j*: end *k*: step
  - jth element is *excluded* from the slice

```
>> b = [2, 3, 5, 7, 11, 13, 17]
>> b[:2]          # get 0th and 1st
[2, 3]
>> b[4:-1]        # last element excluded
[11, 13]
>> b[4:]          # last element included
[11, 13, 17]
>> b[::2]         # skip every other element
[2, 5, 11, 17]
>> b[3::-1]       # reverse list, from 4th element backwards
[7, 5, 3, 2]
```

Relational Operator

- Sequence types are compared *by value*

```
>> b = “hello”
>> b[:5] == “hell”
True
>> a = [1, 2, 3]
>> a > [8, -9]    # lexicographical order
False
```

- Check for alias (compare by *reference*)
  - `is` operator

```
>> a = b = [1, 2, 3]
>> a is b
True
>> c = [1, 2, 3]
```

```
>> a is c
False
>> a == c
True
```

Built-in Functions

- Many operate on *iterables*
- Iterable
  - An object that contains elements you can iterate through
    - Go through each element one after another
  - All sequence types are iterable!
- E.g. sorted – returns a *new* list of sorted elements

```
>> b = [5, 9, 1, 2]
>> sorted(b) # returns a copy
[1, 2, 5, 9]
>> b
[5, 9, 1, 2]
```

```
>> sorted(“bad”)
[‘a’, ‘b’, ‘d’]
>> sorted((3, 2, 1))
[1, 2, 3]
```

Foreach loop

```
>> for n in [2, 3, 5]:
..   print(n+2)
4
5
7
>> for c in “hello”:
..   print(c.upper())
H
E
L
L
O
```

```
# enumerate is a built-in
# function; returns a tuple
>> s = “world”
>> for i, c in enumerate(s):
..   print(“%d: %s”%(i, c))
0: w
1: o
2: r
3: l
4: d
```

Range

- a special sequence type
  - Used to loop fixed number of times
  - Generates numbers from *start* to *end*, with a *step*

```
>> for n in range(1, 6, 2):
..    print(n, end=" ")
1 3 5
```

- *Lazy iterable*
  - computes as you loop through

```
>> x = range(6)
>> x
range(0, 6)
>> list(x)              # turn range into a list
[0, 1, 2, 3, 4, 5]
```

Membership Operator

- Checks for existence of element

```
>> 5 in [3, 6, “5”]
False
>> 5 in [5, “hello”, 3]
True
```

- Check for absence of element

```
>> ‘a’ not in “banana”
False
>> ‘seed’ not in “banana”
True
```

Length Function

```
>> len([1, 2, 3, 4])
4
>> len(“hello”)
5
>> len([])
0

# in Python
import sys                  # arguments to program stored here
argc = len(sys.argv)        # argc (C++) is length of sys.argv

// in C++
int main(int argc, const char * argv[]) {
      …
```

Repetition and Concatenation

```
>> “hello ” * 3
‘hello hello hello ’

>> [0] * 4                  # common used to initialize list
[0, 0, 0, 0]

>> a = “hello”
>> b = “world”
>> a + “ ” + b              # concatenate three strings
‘hello world’

>> [1, 2] + [3, 4]          # concatenate two lists
[1, 2, 3, 4]
```