NPTEL MOOC

PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON

Week 2, Lecture 3

Madhavan Mukund, Chennai Mathematical Institute
http://www.cmi.ac.in/~madhavan

Types of values in Python

- Numbers: `int, float`
  - Arithmetic operations `+,-,*,/,…`
- Logical values: `bool, {True, False}`
  - Logical operations `not,and,…`
  - Comparisons `==,!=,<,>,<=,>=`
- Strings: `str,` sequences of characters
  - Extract by position `s[i]`, slice `s[i:j]`
  - Concatenation `+`, length `len()`, …

Lists

- Sequences of values

```
factors = [1,2,5,10]
names = ["Anand","Charles","Muqsit"]
```

- Type need not be uniform

```
mixed = [3, True, "Yellow"]
```

- Extract values by position, slice, like `str`

`factors[3]` is `10`, `mixed[0:2]` is `[3,True]`

- Length is given by `len()`

`len(names)` is `3`

Lists and strings

- For `str`, both a single position and a slice return strings

```
h = "hello"
h[0] == h[0:1] == "h"
```

- For lists, a single position returns a value, a slice returns a list

```
factors = [1,2,5,10]
factors[0] == 1, factors[0:1] == [1]
```

Nested lists

- Lists can contain other lists

```
nested = [[2,[37]],4,["hello"]]
```

nested[0] is [2,[37]]

nested[1] is 4

nested[2][0][3] is "l"

nested[0][1:2] is [[37]]

Updating lists

- Unlike strings, lists can be updated in place

```
nested = [[2,[37]],4,["hello"]]
nested[1] = 7
```

`nested` is now `[[2,[37]],7,["hello"]]`

```
nested[0][1][0] = 19
```

`nested` is now `[[2,[19]],7,["hello"]]`

- Lists are mutable, unlike strings

Mutable vs immutable

- What happens when we assign names?

```
x = 5
y = x
x = 7
```

- Has the value of y changed?
  - No, why should it?
  - Does assignment copy the value or make both names point to the same value?

Mutable vs immutable …

- Does assignment copy the value or make both names point to the same value?
- For immutable values, we can assume that assignment makes a fresh copy of a value
  - Values of type `int, float, bool, str` are immutable
- Updating one value does not affect the copy

Mutable vs immutable …

- For mutable values, assignment does not make a fresh copy

```
list1 = [1,3,5,7]
list2 = list1
list1[2] = 4
```

- What is `list2[2]` now?
  - `list2[2]` is also `4`
- `list1` and `list2` are two names for the same list

Copying lists

- How can we make a copy of a list?
- A slice creates a new (sub)list from an old one
- Recall `l[:k]` is `l[0:k]`, `l[k:]` is `l[k:len(l)]`
- Omitting both end points gives a full slice

```
l[:] == l[0:len(l)]
```

- To make a copy of a list use a full slice

```
list2 = list1[:]
```

Digression on equality

- Consider the following assignments

```
list1 = [1,3,5,7]
list2 = [1,3,5,7]
list3 = list2
```

- All three lists are equal, but there is a difference
  - `list1` and `list2` are two lists with same value
  - `list2` and `list3` are two names for same list

Digression on equality …

```
list1 = [1,3,5,7]
list2 = [1,3,5,7]
list3 = list2
```

- `x == y` checks if `x` and `y` have same value
- `x is y` checks if `x` and `y` refer to same object

```
list1 == list2 is True
list2 == list3 is True

list2 is list3 is True
list1 is list2 is False
```

Concatenation

- Like strings, lists can be glued together using +

```
list1 = [1,3,5,7]
list2 = [4,5,6,8]
list3 = list1 + list2
```

- `list3` is now `[1,3,5,7,4,5,6,8]`
- Note that + always produces a new list

```
list1 = [1,3,5,7]
list2 = list1
list1 = list1 + [9]
```

- `list1` and `list2` no longer point to the same object

Summary

- Lists are sequences of values
  - Values need not be of uniform type
  - Lists may be nested
- Can access value at a position, or a slice
- Lists are mutable, can update in place
  - Assignment does not copy the value
  - Use full slice to make a copy of a list