Lists

Introduction to Computer Programming

Dr. Paul Vrbik

October 17, 2018

Definition (Tuple)

A tuple is an ordered sequence of elements. An n-tuple is a tuple with exactly n elements.

Round brackets `()` are used to create tuples in Python.

We sometimes call tuples like $(1, 2)$ and $(1, 2, 3)$ couples and triples.

The simplest data structure in Python is the tuple.

```
>>> xs = (1,2)
>>> type(xs)
<class 'tuple'>
```

Tuples (like strings) are immutable

```
>>> xs[0]
1
>>> xs[0] = 2
TypeError: 'tuple' object does not support item
assignment
```

Definition

A list is a mutable tuple.

Square brackets `[ ]` are used to create lists in Python.

```
>>> xs = [1, 2]
>>> type(xs)
<class 'list'>


>>> xs[0]
1
>>> xs[0] = 2*xs[1]
>>> xs[0]
4
```

List Built-Ins

```
>>> dir(list)
['__add__', '__class__', '__contains__', '__delattr__',
'__delitem__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__getitem__', '__gt__',
'__hash__', '__iadd__', '__imul__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__',
'__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__',
'__setattr__', '__setitem__', '__sizeof__', '__str__',
'__subclasshook__', 'append', 'clear', 'copy', 'count',
'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

Append

```
>>> xs = [0, 1, 2]
>>> xs.append(3)
>>> xs
[0, 1, 2, 3]
```

code equivalent to...

```
>>> xs = [0, 1, 2]
>>> xs = xs + [3]
>>> xs
[0, 1, 2, 3]
```

Clear

```
>>> xs = [0, 1, 2]
>>> xs.clear()
>>> xs
[]
```

code equivalent to...

```
>>> xs = [0, 1, 2]
>>> xs = []
```

Copy

```
>>> xs = [1, 2, 3]

>>> ys = xs
>>> ys[-1] = 9

>>> xs
[1, 2, 9]
```

Copy

```
>>> xs = [1, 2, 3]

>>> ys = xs.copy()
>>> ys[-1] = 9

>>> xs
[1, 2, 3]

>>> ys
[1, 2, 9]
```

Count

```
>>> xs = [1, 2, 2, 3, 3, 3]
>>> xs.count(2)
2
>>> xs.count(4)
0


>>> xs = [[1,2,2], [3,3,3]]
>>> xs.count(2)
0
>>> xs.count([3,3,3])
1
```

Extend

```
>>> xs = [1, 2, 3]

>>> xs.append([4,5])
>>> xs
[1, 2, 3, [4,5]]
```

Extend

```
>>> xs = [1, 2, 3]
```

```
>>> xs.extend([4,5])
>>> xs
[1, 2, 3, 4, 5]
```

code equivalent to...

```
>>> xs =  xs + [4, 5]
```

Index

```
>>> xs = ['a', 'b', 'c']
>>> xs.index('c')
2
>>> xs[xs.index('c')]
'b'
>>> xs = ['a', 'b', 'c', 'b', 'b', 'd']
>>> xs.index('b')
1
>>> xs.index('e')
ValueError: 'e' is not in list
```

Insert

```
>>> xs = [0, 1, 2, 3, 4, 5]
>>> len(xs)                                    Length of x
6
>>> xs.insert(-2, 9)
>>> xs
[0, 1, 2, 3, 9, 4, 5]
>>> len(xs)
7
>>> xs.insert(100, 9)
>>> xs
[0, 1, 2, 3, 9, 4, 5, 9]                       note index does not exist
```

Pop

```
>>> xs = [0, 1, 2, 3]


>>> x = xs.pop()
>>> x
3
>>> xs
[0, 1, 2]
```

Pop

```
>>> xs = [0, 1, 2, 3, 4]
>>> while xs:
...     print(xs.pop())

>>> len(xs)
0
```

Remove

```
>>> xs = [0, 1, 5, 2, 3, 5]
>>> xs.remove(5)
>>> xs
[0, 1, 2, 3, 5]
>>> xs.remove(5)
>>> xs
[0, 1, 2, 3]
>>> xs.remove(5)
ValueError: list.remove(x): x not in list
```

Question

Write a function that removes all instances of `x:int` from `xs:List[int]`.

Answer

```
def remove_all(xs:List[int], x:int) -> List[int]
```

Note, in order to say `xs:List[int]` as a type-check we must do

```
from typing import List
```

Reverse

```
>>> xs = [0, 1, 2, 3]
>>> xs.reverse()
>>> xs
[3, 2, 1, 0]
```

code equivalent to...

```
>>> xs = xs[::-1]
```

Sort

```
>>> xs = [1, 0, 8, 3, -2]
>>> xs.sort()
[-2, 0, 1, 3, 8]
```

Note increasing order.

```
>>> xs = [1, 0, 8, 3, -2]
>>> xs.sort(reverse=True)
[8, 3, 1, 0, -2]
```

Note decreasing order.

Comparison

```
>>> [1,2,3] < [4,5,6]
True

>>> [7,2,3] < [4,5,6]
False
```

Point-wise comparisons from position zero.

```
>>> [] < [1]
True
```

Slicing

```
>>> xs = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> xs[3:6]
[3, 4, 5]

>>> xs[::2]
[0, 2, 4, 6, 8]

>>> xs[7:2:-2]
[7, 5, 3]
```

Passing Lists to Functions

It is possible to "unbracket" a list when passing to functions

```
>>> def f(x, y, z):
...     return x + y + z
>>> x = [1, 2, 3]
>>> f(x)
TypeError: f() missing 2 required positional arguments:
'y' and 'z'


>>> f(*x)
6
```

Next Time

1. Looping over lists,
2. `range`,
3. Loop nesting,
4. Nested lists, and
5. List Comprehension.