COMP10001 Foundations of Computing
Tuples and Lists

Semester 2, 2021

Chris Leckie, Marion Zalk and Farah Khan

— VERSION: 1520, DATE: APRIL 4, 2019 —

Lecture Agenda

- Last lecture
  - Functions (cont.)
  - Methods
- This lecture
  - Comments
  - Tuples
  - Lists

Lecture Outline

1. Comments
2. Tuples
3. Lists

Comments

- Comments are notes of explanation that document lines or sections of a program, which follow a # (hash) character
- Python ignores anything following a # on a single line (multi-line “commenting” possible with `"""`):

```
# OK, here goes
"""Three blind mice,
Three blind mice,
..."""
print("Hello world")
```

Commenting Expectations

- For this subject we require:
  - All key variables should have comments about what they are used for (as should user-defined functions)
  - Your code should describe **why** you do things, not **what** you do
  - Commenting can also be used to stop lines of code from being executed. This is called "commenting out" code.

Lecture Outline

Keeping it together: Tuples

- Tuples are just like strings but:
  - each element can be something other than a character
  - we use (,) rather than " " to build them

```
>>> costs = (1, 2.6, 7.1, -3.14)
>>> print(costs[0])
1
>>> print(costs[2:4])
(7.1, -3.14)
```

When would I Use Tuples?

- Representing “multi-variate” objects:
    - representing coordinates `(x, y, z)`
    - health records `(name, address, ...)`
    - playing cards `(value, suit)`
    - map positions `(latitude, longitude)`
    - mental state `(love, hate, desire, beliefs, ...)`
    - limb positions `(angle, voltage, resistance)`

Useful Coding Applications of Tuples

- To return multiple values:
  ```
  return (name, age, gender)
  ```
- To swap values between variables:
  ```
  (a, b) = (b, a)
  ```
- To test for one of a series of values:
  ```
  number in (12, 1, 2)
  ```
- As keys to dictionaries (see later ...)

Just like Strings, Tuples are “Immutable”

- Once they are created, you cannot change elements

```
>>> data = (1, True, 'alice', 'bob')
>>> data[0] = 0
TypeError: 'tuple' object does not support ...
>>> data = "Alice and Bob"
>>> data[0] = 'H'
TypeError: 'str' object does not support ...
```

Variable-arity Functions: Redux

- A second way of defining a "variable-arity" function is by identifying a parameter as generating a variable-sized tuple of any "leftover" arguments:

```python
def varfun(num, *rest):
    return (num, rest)
```

```
>>> varfun(1, 2)
(1, (2,))
>>> varfun(1)
(1, ())
>>> varfun(1, 2, 3)
(1, (2, 3))
```

Lecture Outline

Lists: Mutable Data Type

- Lists are just like tuples but:
  - they are mutable
  - we use [,] rather than (,) to build them

```
>>> ["head", "tail", "tail"]
>>> [5, 5, 30, 10, 50]
>>> [1, 2, "buckle my shoe", 3.0, 4.0]
```

- As with all types, we can assign a list to a variable:

```
>>> fruit = ["orange", "apple", "apple"]
```

List Indexing and Splitting

- To access the items in a list we can use indexing (just like we do with strings and tuples):

```
>>> stuff_list = ["12", 23, 4, 'burp']
>>> stuff_list[-1]
'burp'
```

- We can similarly slice a list:

```
>>> stuff_list[:2]
['12', 23]
```

and calculate the length of a list with `len`

```
>>> len(stuff_list)
4
```

List Mutation

- Unlike tuples and strings, we can modify the internals of lists directly, either via assignment:

```
>>> stuff_list = ["12", 23, 4, 'burp']
>>> stuff_list[0] = '21'
>>> stuff_list
['21', 23, 4, 'burp']
>>> stuff_list[:2] = ['boing!']
>>> stuff_list
['boing!', 4, 'burp']
```

- ... or via methods that modify the internals of the list ...

List Mutation

- `append()` = add (single) item to end of list

```
>>> lst = [1, 2, 3]
>>> lst.append(4)
>>> lst
[1, 2, 3, 4]
```

- `remove()` = remove the first instance of a given value

```
>>> lst.remove(1)
>>> lst
[2, 3, 4]
```

List Mutation

- `pop()` = remove the element of the given index given value

```
>>> lst = [1, 2, 3]
>>> a = lst.pop(2)
>>> b = lst.pop(0)
>>> a, b, lst
(3, 1, [2])
```

- `sort()` = sort the contents of the list (in-place)

```
>>> lst = [4, 1, 3, 2]
>>> lst.sort()
>>> lst
[1, 2, 3, 4]
```

But What's the Difference?

- It seems that tuples and lists are the same, why have both?
- Important difference: mutability

```
>>> mylist = [1,2,3]
>>> mytuple = (1,2,3)
>>> mylist[1] = 6 ; print(mylist)
[1,6,3]
>>> mytuple[1] = 6 ; print(mytuple)
TypeError: 'tuple' object does not support ...
```

Mutability

- Types in Python can be either:
  - “immutable”: the state of objects of that type cannot be changed after they are created
  - “mutable”: the state of objects of that type **can** be changed after they are created
- Quiz:
  - Are strings mutable?
  - Are ints and floats mutable?

Lecture Summary

- What is a tuple?
- What is a list?
- What are mutable types?