COMP10001 Foundations of Computing
Tuples and Iteration

Semester 1, 2019
Tim Baldwin, Nic Geard, Farah Khan, and Marion Zalk

THE UNIVERSITY OF MELBOURNE

— version: 1493, date: March 26, 2019 —

Lecture Agenda

- This lecture — Grok Worksheet 5
  - Functions (cont.)
  - Methods
  - Comments
  - Tuples
- This lecture – Grok Worksheet 8
  - Project 1
  - Tuples (cont.)
  - Iteration

Lecture Outline

Project 1: Context

- **Social network analysis**, strongly themed around making sense of how a company like Cambridge Analytica was potentially able to influence people's voting behaviour, based around the key elements of:
  - network degree/propagation
  - homophily ("birds of a feather flock together")
- For our purposes, in a social network each individual is a "node", and a friendship link is an (undirected) "edge" between a pair of nodes

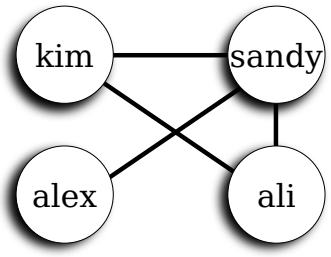
Project 1: Context

- The Cambridge Analytica scandal:
  - 270k Facebook users accepted payment to play a personality test game called "This is your Digital Life"
  - under the T&Cs of the game, the full history of posts, user profile information, newsfeed, and list of friends of each of those users was crawled *along with all this data for each of their friends*
  - the combined dataset this resulted in totalled around 87m distinct users (around 5% of all Facebook users)
  - this data was then used commercially for targeted advertising in various contexts, including the most recent US presidential election

Project 1: Context

- We will represent a social network as a sequence of edges (= friendship relations) over pairs of individuals, e.g.:

```
[('kim', 'sandy'), ('sandy', 'alex'), \
 ('ali', 'kim'), ('ali', 'sandy')]
```



Project 1: Key Concepts

- The **distance** between two nodes (= individuals) is defined by the shortest path between them:

$$\begin{aligned} \text{distance}(\text{kim}, \text{alex}) &= 2 \\ \text{distance}(\text{kim}, \text{sandy}) &= 1 \\ \text{distance}(\text{kim}, \text{kim}) &= 0 \end{aligned}$$

Project 1: Key Concepts

- The **besties** of user x is the set $\{y_i : \text{distance}(x, y_i) = 1\}$

```
>>> friend_besties('kim', {'kim': {'sandy', 'ali'}, ...
['ali', 'sandy']
>>> friend_besties('alex', {'kim': {'sandy', 'ali'}, ..
['sandy']
```


Project 1: Key Concepts

- The **second-besties** of user x is the set $\{y_i : \text{distance}(x, y_i) = 2\}$

```
>>> friend_second_besties('kim', {'kim': {...})
['alex']
>>> friend_second_besties('alex', {'kim': {...})
['ali', 'kim']
```


Lecture Outline

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:

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

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


Lecture Outline

Iteration

- Iteration = one final, essential tool to make the computer do something over and over again
  - Repeat something forever
  - Repeat something until something happens
  - Repeat something a fixed number of times
    - move turtle one pixel to the right 10 times over
    - print a copy 7 times over
    - play Nyan Cat 15 times

Iteration: `for` Loops

- A loop over sequences, usually in one of two forms:
  - via a numeric range generated by `range()`
  - indirectly over the elements of a sequence
- The general idea is that we work our way through (all of) an "iterable" (`str` for our current purposes) of items one item at a time, in sequence

Iteration: `for` Loops

- Basic form:

```
for␣<var>␣in␣<iterable>:
␣␣␣␣statement␣block
```

- Note: `in` here is not (quite) the same as the comparison operator of the same name
- It is called `for...in...` because of the "for all" ($\forall$) quantifier

A Useful Function for creating a sequence

- `range(start=0,end,step=1)`: generate a sequence of `int` values from `start` (inclusive) to `end` (non-inclusive), counting `step` at a time

```
>>> for i in range(5):
...     print(i, end=" ")
0 1 2 3 4
>>> for i in range(0, 10, 2):
...     print(i, end=" ")
0 2 4 6 8
>>> for i in range(10, 0, -1):
...     print(i, end=" ")
10 9 8 7 6 5 4 3 2 1
```

Simple Examples

```
mylen = 0
for i in "abc":
    mylen = mylen + 1
print(mylen)
```

is equivalent to:

```
mylen = 0
mylen = mylen + 1  # i = 'a'
mylen = mylen + 1  # i = 'b'
mylen = mylen + 1  # i = 'c'
print(mylen)
```


Simple Examples

- A more interesting example:

```
vowels = 0
word = "rhythm"
for char in  word:
    if char in "aeiou":
        vowels = vowels + 1
print(vowels)
```


Simple Examples

- Or alternatively with `range()`:

```
vowels = 0
word = "rhythm"
for i in range(len(word)):
    if word[i] in "aeiou":
        vowels = vowels + 1
print(vowels)
```

Particularly useful when you need to be able to recover *where* matches occurred

Lecture Summary

- What is a list?
- What are mutable types?
- What is a `for` loop?