Introduction to Python Programming

Kyle Jamieson

Networked Computer Systems Programme
Department of Computer Science
University College London
Setup and startup

% bash
$ cd ~
$ cp ~jamieson/pyintro_distribution.tgz .
$ gunzip pyintro_distribution.tgz
$ tar xvpf pyintro_distribution.tar
$ cd pyintro/src

• To start the Python interpreter, use python_wrapper:

$ python-wrapper
Python 2.6.2 (r262:71600, Sep 2 2009, 18:21:20)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-44)] on linux2
Type "help", "copyright", "credits" or "license" for
more information.
>>> 

• To exit the interpreter, type Ctrl+D
• Can type any Python expression at the primary command prompt (>>>)


Running Python scripts

• Start a text editor (a simple editor gedit, or your favorite), edit and save hello.py in tubelab/src

  Contents of file hello.py:

  ```
  $ cd pyintro/src
  $ gedit hello.py
  ```

  ```
  print "Hello, world."
  ```

• Run your first program!

  ```
  $ ./python-wrapper hello.py
  Hello, world.
  $ 
  ```

• Now, let’s start the Python interpreter and explore the language, starting with basic expressions and types
For further information

• Python tutorial
  – http://docs.python.org/tutorial

• Library reference
  – http://docs.python.org/library/index.html

• Language reference
  – http://docs.python.org/reference/index.html
TubeLab

- **Objective:** Print out directions between pair of Zone 1 stations
  - Directions should correspond to shortest distance between the two stations, measured by sum of station stops and transfers between platforms
  - Represent this map with Python data structures
  - Breadth-first search to find shortest routes

http://www.tfl.gov.uk/assets/downloads/standard-tube-map.gif
A user’s interaction with TubeLab

Origin station: Paddington
Destination station: Leicester Square
To get from Paddington station to Leicester Square station
Begin at the Paddington station Bakerloo line platform
Take the Bakerloo Line 6 stop(s) to Piccadilly Circus station
Transfer at Piccadilly Circus station to the Piccadilly line platform
Take the Piccadilly Line 1 stop(s) to Leicester Square station

Origin station: Green Park
Destination station: Liverpool Street
To get from Green Park station to Liverpool Street station
Begin at the Green Park station Jubilee line platform
Take the Jubilee Line 4 stop(s) to London Bridge station
Take the Northern Line 1 stop(s) to Bank station
Transfer at Bank station to the Central line platform
Take the Central Line 1 stop(s) to Liverpool Street station

Origin station: ^D
Tada!

[jamieson@shannon:src] $
Two key data types

• *Station*: named on the tube map, contains platforms
  – Example: Euston Station has three platforms:
    • Two platforms on the *Northern* Line
    • One platform on the *Victoria* line
  – Example: Tottenham Court Road Station has one platform on the *Northern* line, one on the *Central* line
  – Example: Great Portland Street Station has one platform on the *Hammersmith and City*, *Circle*, and *Metropolitan* lines

• *Platform*: associated with one or more lines
• *Line*: represent with Python strings
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph \( G = (V, E) \) and source vertex \( s \)
- **Output:** A breadth-first tree with root \( s \), containing shortest paths between \( s \) and any other vertex

1. BFS\((G, s)\):
2. \( \pi(s) \leftarrow \text{None} \)
3. \( Q \leftarrow \{s\} \)
4. Visited \( \leftarrow \{\} \)
5. **while** \( Q \neq \{\} \):
6. \( u \leftarrow \text{Q.dequeue()} \)
7. Visited.add\((u)\)
8. **foreach** \( v \in \text{Adj}(u) \):
9. **if** \( v \notin \text{Visited} \) and \( v \notin Q \):
10. \( Q.\text{enqueue}(v) \)
11. \( \pi(v) \leftarrow u \)
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breath-first tree with root $s$, containing shortest paths between $s$ and any other vertex

```
1 BFS(G, s):
2 π(s) ← None
3 Q ← {s}
4 Visited ← {s}
5 while Q ≠ {}:
6   u ← Q.dequeue()
7   Visited.add(u)
8   foreach v ∈ Adj(u):
9     if v ∉ Visited and v ∉ Q:
10        Q.enqueue(v)
11        π(v) ← u
```
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breath-first tree with root $s$, containing shortest paths between $s$ and any other vertex

```python
1 BFS(G, s):
2 $\pi(s) \leftarrow \text{None}$
3 $Q \leftarrow \{s\}$
4 Visited $\leftarrow \{s\}$
5 while $Q \neq \{}$
6 \hspace{1em} $u \leftarrow Q.\text{dequeue}()$
7 \hspace{1em} Visited.add($u$)
8 \hspace{1em} foreach $v \in \text{Adj}(u)$:
9 \hspace{2em} if $v \not\in$ Visited and $v \not\in Q$:
10 \hspace{3em} $Q.\text{enqueue}(v)$
11 \hspace{2em} $\pi(v) \leftarrow u$
```
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breadth-first tree with root $s$, containing shortest paths between $s$ and any other vertex

```
1 BFS(G, s):
2 \( \pi(s) \leftarrow \text{None} \)
3 \( Q \leftarrow \{s\} \)
4 \( \text{Visited} \leftarrow \{s\} \)
5 \textbf{while } Q \neq \{\}:
6 \quad u \leftarrow Q.\text{dequeue}()
7 \quad \text{Visited}.\text{add}(u)
8 \quad \textbf{foreach } v \in \text{Adj}(u):
9 \quad \quad \textbf{if } v \notin \text{Visited} \text{ and } v \notin Q:
10 \quad \quad \quad Q.\text{enqueue}(v)
11 \quad \quad \pi(v) \leftarrow u
```
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breath-first tree with root $s$, containing shortest paths between $s$ and any other vertex

```
1 BFS(G, s):
2   \( \pi(s) \leftarrow \text{None} \)
3   \( Q \leftarrow \{s\} \)
4   \( \text{Visited} \leftarrow \{s\} \)
5   \textbf{while} \( Q \neq \{} \):
6     \( u \leftarrow Q.\text{dequeue}() \)
7     \( \text{Visited}.\text{add}(u) \)
8     \textbf{foreach} \( v \in \text{Adj}(u) : \)
9         \textbf{if} \( v \not\in \text{Visited} \text{ and } v \not\in Q : \)
10            \( Q.\text{enqueue}(v) \)
11            \( \pi(v) \leftarrow u \)
```
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breath-first tree with root $s$, containing shortest paths between $s$ and any other vertex

```python
1 BFS(G, s):
2 $\pi(s) \leftarrow \text{None}$
3 $Q \leftarrow \{s\}$
4 Visited $\leftarrow \{s\}$
5 while $Q \neq \{\}$:
6 $u \leftarrow Q$.dequeue()
7 Visited.add($u$)
8 foreach $v \in \text{Adj}(u)$:
9   if $v \notin \text{Visited}$ and $v \notin Q$:
10      $Q$.enqueue($v$)
11      $\pi(v) \leftarrow u$
```
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth.
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breadth-first tree with root $s$, containing shortest paths between $s$ and any other vertex.

1. BFS($G$, $s$):
2. $\pi(s) \leftarrow \text{None}$
3. $Q \leftarrow \{s\}$
4. $\text{Visited} \leftarrow \{s\}$
5. **while** $Q \neq \{\}$:
6. $u \leftarrow Q$.dequeue()
7. $\text{Visited}.add(u)$
8. **foreach** $v \in \text{Adj}(u)$:
9. **if** $v \notin \text{Visited}$ and $v \notin Q$:
10. $Q$.enqueue($v$)
11. $\pi(v) \leftarrow u$
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breath-first tree with root $s$, containing shortest paths between $s$ and any other vertex

```
1 BFS(G, s):
2 \( \pi(s) \leftarrow \text{None} \)
3 \( Q \leftarrow \{s\} \)
4 Visited \( \leftarrow \{s\} \)
5 \text{while } Q \neq \emptyset:\
6 \quad u \leftarrow Q\.dequeue() \)
7 \quad Visited\.add(u) \)
8 \text{foreach } v \in \text{Adj}(u): \)
9 \quad \text{if } v \notin \text{Visited and } v \notin Q:\
10 \quad \quad Q\.enqueue(v) \)
11 \quad \pi(v) \leftarrow u \)
```
Breadth-first search (BFS)

- Explore graph, expanding frontier between undiscovered and discovered vertices uniformly across its breadth
- **Input:** Undirected graph $G = (V, E)$ and source vertex $s$
- **Output:** A breadth-first tree with root $s$, containing shortest paths between $s$ and any other vertex

1. $\text{BFS}(G, s)$:
2. $\pi(s) \leftarrow \text{None}$
3. $Q \leftarrow \{s\}$
4. $\text{Visited} \leftarrow \{s\}$
5. **while** $Q \neq \{}$
6. \hspace{1em} $u \leftarrow Q.\text{dequeue}()$
7. \hspace{1em} $\text{Visited}.\text{add}(u)$
8. \hspace{1em} **foreach** $v \in \text{Adj}(u)$:
9. \hspace{2em} **if** $v \notin \text{Visited}$ and $v \notin Q$:
10. \hspace{2.5em} $Q.\text{enqueue}(v)$
11. \hspace{2em} $\pi(v) \leftarrow u$
From tube map to graph representation

• Each vertex in the abstract graph corresponds to a Platform
• Each edge is either a transfer or a trip between platforms of different stations on the same line
• Examples:
  – (camden, euston_b): Northern line south one stop
  – (euston_b, euston_c): Transfer at Euston Station to the Victoria line
• BFS happens in this graph, breadth-first tree constructed using Platform.set_predecessor(Platform)
Code walkthrough: Stations

- A Station groups a number of platforms together
- The iterator returned by platforms(self) yields all Platforms contained within the Station

```python
class Station:
    def __init__(self, name, platforms):  # list of platforms
        pass
    def __str__(self):  # return a string representation
        pass
    def platforms(self):  # returns a generator == platform
        pass
```

• A Station groups a number of platforms together
• The iterator returned by platforms(self) yields all Platforms contained within the Station
Code walkthrough: Platforms

platform.py

class Platform
    def __init__(self, lines) # list of strings: which lines?
    def set_station(self, station) # set containing station
    def add_neighbor(self, line, other_platform)
    def set_predecessor(self, platform) # for BFS
    def set_predecessor_line(self, line) # for BFS
    def neighbors(self) # returns an iterator ==> (platform, line)

• Convention: the iterator returned by neighbors(self) yields (Platform, None) for transfers within a Station
Code walkthrough: Putting it together

tubedata.py

```python
from platform import *
from station import *
northern = 'Northern'
victoria = 'Victoria'
central = 'Central'
tott_ct_rd_a = Platform([northern])
tott_ct_rd_b = Platform([central])
tott_ct_rd_sta = Station('Tottenham Court Road', [tott_ct_rd_a, hott_ct_rd_b])
goodge_st = Platform([northern])
goodge_st_sta = Station('Goodge Street', [goodge_st])
warren_st = Platform([northern, victoria])
warren_st_sta = Station('Warren Street', [warren_st])
euston_a = Platform([northern])
euston_b = Platform([northern])
euston_c = Platform([victoria])
euston_sta = Station('Euston', [euston_a, euston_b, euston_c])
connect(northern, [waterloo_b, embankment_a, charing_cross_a, leicester_sg, hott_ct_rd_a, goodge_st, warren_st, euston_a, mornington_crescent, camden_town])
```

**Exercise:** fire up Python, and print all neighbors of the Victoria line platform in Euston Station (euston_c):

```bash
$ cd tubelab
$ python-wrapper
```
Code walkthru: Main program

- The program entry point, command line interface (CLI), and shortest-paths calculator are in file `tubelab.py`
  - `bfs_directions`: computes shortest-paths with BFS
  - `completer`: allows user to use tab to complete
  - Entry point: runs a while loop until Ctrl+D or Ctrl+C input from user, takes input, passes it to `bfs_directions`

**Main lab assignment**

Implement BFS’s “inner loop” in `bfs_directions`.

- To run/test:
  
  $ python-wrapper tubelab.py