



# Tutorial: Large Scale Network Analytics with SNAP

<http://snap.stanford.edu/proj/snap-www>

Rok Sosič, Jure Leskovec  
Stanford University





# Snap.py: SNAP for Python

Rok Sosič, Jure Leskovec  
Stanford University

# What is Snap.py ?

- **Snap.py** (pronounced “snappy”):  
**SNAP for Python**

<http://snap.stanford.edu/snappy>

User interaction	Python
Snap.py	Python
SNAP	C++

Solution	Fast Execution	Easy to use, interactive
C++	✓	
Python		✓
Snap.py (C++, Python)	✓	✓

# Installing Snap.py

- **Requires Python 2.x**
  - Download and install Python 2.x:  
<http://www.python.org>
- **Download the Snap.py for your platform:**  
<http://snap.stanford.edu/snappy>
  - Packages for Mac OS X, Windows, Linux (CentOS)
    - OS must be 64-bit
    - Mac OS X, 10.7.5 or later
    - Windows, install Visual C++ Redistributable Runtime  
<http://www.microsoft.com/en-us/download/details.aspx?id=30679>
- **Installation:**
  - Follow instructions on the Snap.py webpage  
**`python setup.py install`**

If you encounter problems, please report them to us or post to the mailing list

# Snap.py: Important

- The most important step:

**Import the snap module!**

```
$ python
```

```
>>> import snap
```

# Snap.py Tutorial

- **On the Web:**

<http://snap.stanford.edu/snappy/doc/tutorial/index-tut.html>

- **We will cover:**

- Basic Snap.py data types
- Vectors, hash tables and pairs
- Graphs and networks
- Graph creation
- Adding and traversing nodes and edges
- Saving and loading graphs
- Plotting and visualization

# Snap.py Naming Conventions (1)

## Variable types/names:

- ...**Int**: an **integer** operation, variable: **GetValInt()**
- ...**Flt**: a **floating** point operation, variable; **GetValFlt()**
- ...**Str**: a **string** operation, variable; **GetDateStr()**

## Classes vs. Graph Objects:

- T...: a **class type**; **TUNGraph**
- P...: type of a **graph object**; **PUNGraph**

## Data Structures:

- ...**V**: a **vector**, variable **TIntV InNIdV**
- ...**VV**: a vector of vectors (i.e., a matrix), variable **FltVV**  
    **TFltVV** ... a matrix of floating point elements
- ...**H**: a **hash table**, variable **NodeH**  
    **TIntStrH** ... a hash table with **TInt** keys, **TStr** values
- ...**HH**: a hash of hashes, variable **NodeHH**  
    **TIntIntHH** ... a hash table with **TInt** key 1 and **TInt** key 2
- ...**Pr**: a **pair**; type **TIntPr**

# Snap.py Naming Conventions (2)

- **Get...:** an **access** method, **GetDeg()**
- **Set...:** a **set** method, **SetXYLabel()**
- **...I:** an **iterator**, **NodeI**
- **Id:** an **identifier**, **GetUId()**
- **NId:** a **node identifier**, **GetNId()**
- **EId:** an **edge identifier**, **GetEId()**
- **Nbr:** a **neighbor**, **GetNbrNId()**
- **Deg:** a **node degree**, **GetOutDeg()**
- **Src:** a **source node**, **GetSrcNId()**
- **Dst:** a **destination node**, **GetDstNId()**



# Basic Types in Snap.py (and SNAP)

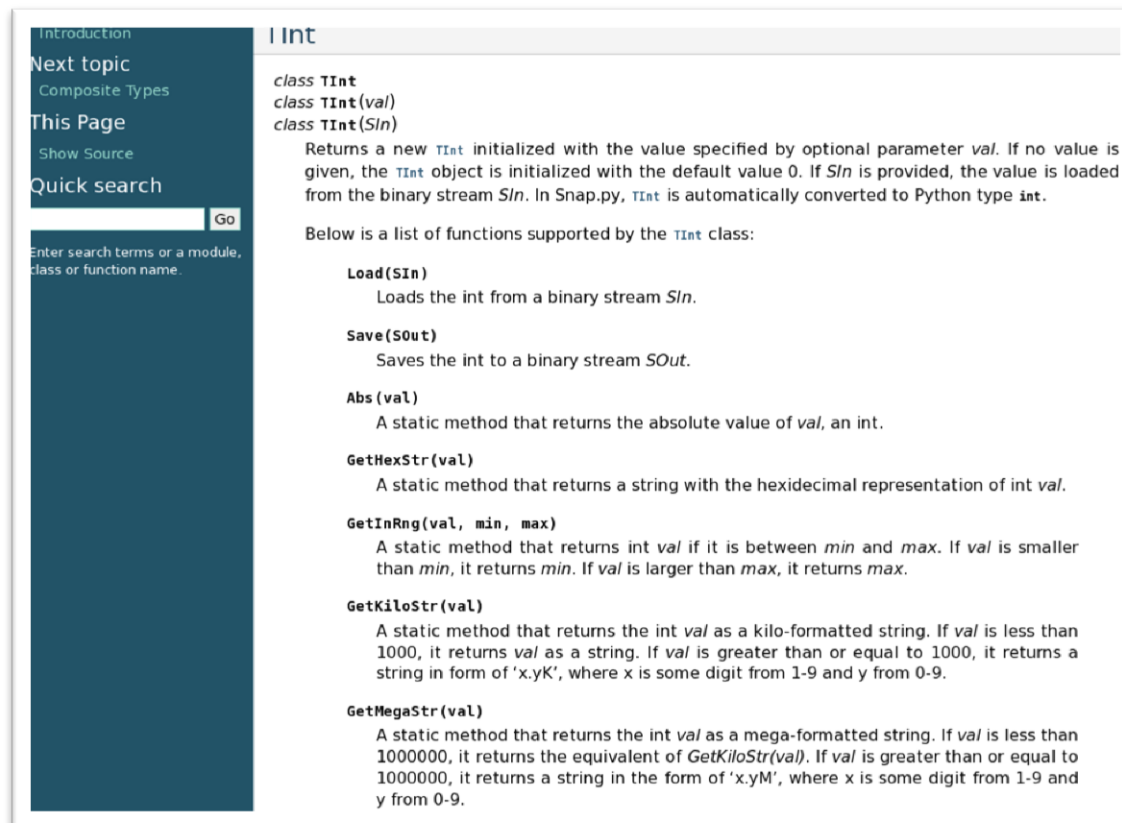
- **TInt**: Integer
- **TFlt**: Float
- **TStr**: String
- Used primarily for constructing composite types
- In general no need to deal with the basic types explicitly
  - Data types are automatically converted between C++ and Python
  - An illustration of explicit manipulation:

```
>>> i = snap.TInt(10)
>>> print i.Val
10
```
- **Note:** do not use an empty string “” in TStr parameters

# Snap.py Reference Documentation

For more information check out Snap.py Reference Manual

<http://snap.stanford.edu/snappy/doc/reference/index-ref.html>



The screenshot shows the Snap.py Reference Documentation page for the `TInt` class. The page has a dark blue sidebar on the left with navigation links: "Introduction", "Next topic" (Composite Types), "This Page" (Show Source), and "Quick search". The main content area is white and titled "TInt". It lists the class definitions: `class TInt`, `class TInt(val)`, and `class TInt(SIn)`. Below this, it explains that `TInt` is initialized with an optional parameter `val`, or the default value 0, or loaded from a binary stream `SIn`. It then lists functions supported by the `TInt` class: `Load(SIn)`, `Save(SOut)`, `Abs(val)`, `GetHexStr(val)`, `GetInRng(val, min, max)`, `GetKiloStr(val)`, and `GetMegaStr(val)`, each with a brief description of its functionality.

Introduction

Next topic  
Composite Types

This Page  
Show Source

Quick search  
Go

Enter search terms or a module, class or function name.

## TInt

```
class TInt
class TInt(val)
class TInt(SIn)
```

Returns a new `TInt` initialized with the value specified by optional parameter `val`. If no value is given, the `TInt` object is initialized with the default value 0. If `SIn` is provided, the value is loaded from the binary stream `SIn`. In Snap.py, `TInt` is automatically converted to Python type `int`.

Below is a list of functions supported by the `TInt` class:

**Load(SIn)**  
Loads the int from a binary stream `SIn`.

**Save(SOut)**  
Saves the int to a binary stream `SOut`.

**Abs(val)**  
A static method that returns the absolute value of `val`, an int.

**GetHexStr(val)**  
A static method that returns a string with the hexadecimal representation of int `val`.

**GetInRng(val, min, max)**  
A static method that returns int `val` if it is between `min` and `max`. If `val` is smaller than `min`, it returns `min`. If `val` is larger than `max`, it returns `max`.

**GetKiloStr(val)**  
A static method that returns the int `val` as a kilo-formatted string. If `val` is less than 1000, it returns `val` as a string. If `val` is greater than or equal to 1000, it returns a string in form of 'x.yK', where x is some digit from 1-9 and y from 0-9.

**GetMegaStr(val)**  
A static method that returns the int `val` as a mega-formatted string. If `val` is less than 1000000, it returns the equivalent of `GetKiloStr(val)`. If `val` is greater than or equal to 1000000, it returns a string in the form of 'x.yM', where x is some digit from 1-9 and y from 0-9.

# SNAP C++ Documentation

## SNAP User Reference Manual

<http://snap.stanford.edu/snap/doc.html>

SNAP Library 2.4, User Reference 2015-05-11 19:40:56  
SNAP, a general purpose, high performance system for analysis and manipulation of large networks

Main Page | Namespaces | **Classes** | Files

Class List | Class Index | Class Hierarchy | Class Members

Search

### TNGraph Class Reference

Classes | Public Types | Public Member Functions | Static Public Member Functions | Private Member Functions | Private Attributes | Friends | List of all members

Directed graph. [More...](#)

```
#include <graph.h>
```

#### Classes

class **TEdgeI**  
Edge iterator. Only forward iteration (operator++) is supported. [More...](#)

class **TNode**

class **TNodeI**  
Node iterator. Only forward iteration (operator++) is supported. [More...](#)

#### Public Types

```
typedef TNGraph TNet
```

```
typedef TPT< TNGraph > PNet
```

#### Public Member Functions

**TNGraph** ()

**TNGraph** (const int &Nodes, const int &Edges)  
Constructor that reserves enough memory for a graph of Nodes nodes and Edges edges. [More...](#)

**TNGraph** (const **TNGraph** &Graph)

**TNGraph** (**TSIn** &SIn)

- TMd5
- TMd5Sig
- TMem
- TMemIn
- TMemOut
- TMin
- TMom
- TMOut
- TNativeCallbackNotify
- TNcpGraphsBase
- TNEANet
- TNEGraph
- TNEq
- TNetConstraint
- TNetIntBs
- TNGraph**
- TNGraphMtx
- TNIBs
- TNodeEDatNet
- TNodeEdgeNet
- TNodeInfo
- TNodeNet
- TNotify
- TNSEException
- TNullNotify
- TNumericalStuff
- TOLx

# Vector Types

- **Sequences of values of the same type**
  - New values can be added at the end
  - Existing values can be accessed or changed
- **Naming convention:  $T\langle\text{value\_type}\rangle V$** 
  - Examples:  $TIntV$ ,  $TFltV$ ,  $TStrV$
- **Common operations:**
  - **Add( $\langle\text{value}\rangle$ ):** append a value at the end
  - **Len():** vector size
  - **[ $\langle\text{index}\rangle$ ]:** get or set a value of an existing element
  - **for i in V:** iteration over the elements

# Vector Example

```
v = snap.TIntV()
```

Create an empty vector

```
v.Add(1)
```

```
v.Add(2)
```

```
v.Add(3)
```

```
v.Add(4)
```

```
v.Add(5)
```

Add elements

```
print v.Len()
```

Print vector size

```
print v[3]
```

```
v[3] = 2*v[2]
```

```
print v[3]
```

Get and set element value

```
for item in v:
```

```
    print item
```

```
for i in range(0, v.Len()):
```

```
    print i, v[i]
```

Print vector elements

# Hash Table Types

- **A set of (key, value) pairs**
  - Keys must be of the same types
  - Values must be of the same type
    - Value type can be different from the key type
  - New (key, value) pairs can be added
  - Existing values can be accessed or changed via a key
- **Naming:  $T\langle\text{key\_type}\rangle\langle\text{value\_type}\rangle H$** 
  - **Examples:**  $TIntStrH$ ,  $TIntFltH$ ,  $TStrIntH$
- **Common operations:**
  - $[\langle\text{key}\rangle]$ : add a new value or get or set an existing value
  - $Len()$ : hash table size
  - **for**  $k$  **in**  $H$ : iteration over keys

# Hash Table Example

```
h = snap.TIntStrH()
```

Create an empty table

```
h[5] = "apple"
```

```
h[3] = "tomato"
```

```
h[9] = "orange"
```

```
h[6] = "banana"
```

```
h[1] = "apricot"
```

Add elements

```
print h.Len()
```

Print table size

```
print "h[3] =", h[3]
```

Get element value

```
h[3] = "peach"
```

```
print "h[3] =", h[3]
```

Set element value

```
for key in h:  
    print key, h[key]
```

Print table elements

# Hash Tables: KeyID

- **T<key\_type><value\_type>H**
  - **Key:** item key, provided by the caller
  - **Value:** item value, provided by the caller
  - **KeyId:** integer, unique slot in the table, calculated by SNAP

KeyId	0	2	5
Key	100	89	95
Value	"David"	"Ann"	"Jason"



# Pair Types

- **A pair of (value1, value2)**
  - Two values
    - type of value1 could be different from the value2 type
  - Existing values can be accessed
- **Naming:  $T\langle\text{type1}\rangle\langle\text{type2}\rangle\text{Pr}$** 
  - **Examples:** TIntStrPr, TIntFltPr, TStrIntPr
- **Common operations:**
  - **GetVal1:** get value1
  - **GetVal2:** get value2

# Pair Example

```
>>> p = snap.TIntStrPr(1, "one")
```

Create a pair

```
>>> print p.GetVal1()
```

```
1
```

Print pair values

```
>>> print p.GetVal2()
```

```
one
```

- **TIntStrPrV**: a vector of (integer, string) pairs
- **TIntPrV**: a vector of (integer, integer) pairs
- **TIntPrFltH**: a hash table with (integer, integer) pair keys and float values

# Basic Graph and Network Classes

- **Graphs vs. Networks Classes:**
  - **TUNGraph**: undirected graph
  - **TNGraph**: directed graph
  - **TNEANet**: multigraph with attributes on nodes and edges
- Object types start with **P...**, since they use wrapper classes for garbage collection
  - **PUNGraph**, **PNGraph**, **PNEANet**
- **Guideline**
  - For class methods (functions) use **T**
  - For object instances (variables) use **P**

# Graph Creation

```
G1 = snap.TNGraph.New()
```

Directed  
graph

```
G1.AddNode(1)
```

```
G1.AddNode(5)
```

```
G1.AddNode(12)
```

Add nodes  
before adding  
edges

```
G1.AddEdge(1,5)
```

```
G1.AddEdge(5,1)
```

```
G1.AddEdge(5,12)
```



G1

```
G2 = snap.TUNGraph.New()
```

Undirected graph,  
directed network

```
N1 = snap.TNEANet.New()
```

# Graph Traversal

## Node traversal

```
for NI in G1.Nodes():  
    print "node id %d, out-degree %d, in-degree %d"  
        % (NI.GetId(), NI.GetOutDeg(), NI.GetInDeg())
```

## Edge traversal

```
for EI in G1.Edges():  
    print "(%d, %d)" % (EI.GetSrcNId(), EI.GetDstNId())
```

## Edge traversal by nodes

```
for NI in G1.Nodes():  
    for DstNId in NI.GetOutEdges():  
        print "(%d %d)" % (NI.GetId(), DstNId)
```

# Graph Saving and Loading

Save text

```
snap.SaveEdgeList(G4, "test.txt", "List of edges")
```

Load text

```
G5 = snap.LoadEdgeList(snap.PNGraph, "test.txt", 0, 1)
```

```
FOut = snap.TFOut("test.graph")  
G2.Save(FOut)  
FOut.Flush()
```

Save binary

```
FIn = snap.TFIn("test.graph")  
G4 = snap.TNGraph.Load(FIn)
```

Load binary

# Text File Format

## ■ Example file: **wiki-Vote.txt**

- Download from <http://snap.stanford.edu/data>

```
# Directed graph: wiki-Vote.txt
```

```
# Nodes: 7115 Edges: 103689
```

```
# FromNodeId    ToNodeId
```

```
0              1
```

```
0              2
```

```
0              3
```

```
0              4
```

```
0              5
```

```
2              6
```

```
...
```

Load text

```
G5 = snap.LoadEdgeList(snap.PNGraph, "test.txt", 0, 1)
```

# Plotting in Snap.py

- **Plotting graph properties**
  - Gnuplot: <http://www.gnuplot.info>
- **Visualizing graphs**
  - Graphviz: <http://www.graphviz.org>
- **Other options**
  - Matplotlib: <http://www.matplotlib.org>



# Plotting with Snap.py

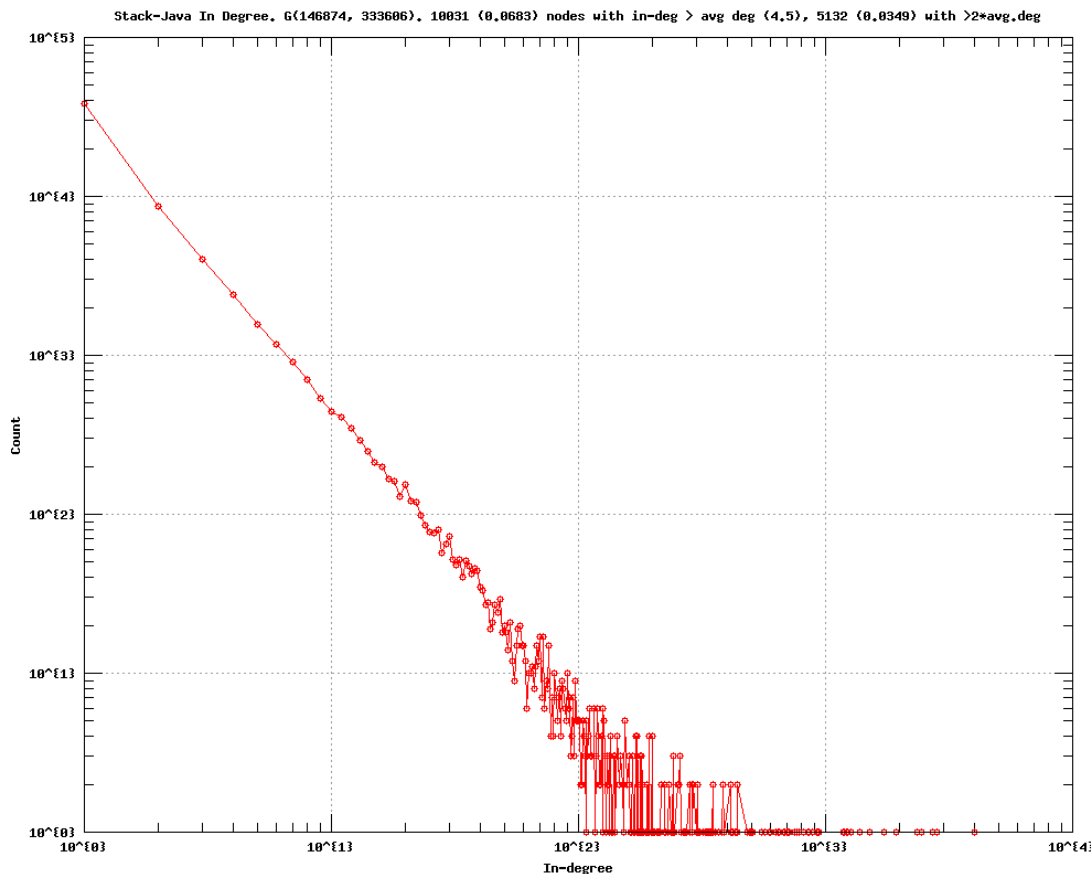
- **Install Gnuplot:**

<http://www.gnuplot.info/>

- Make sure that the directory containing wgnuplot.exe (for Windows) or gnuplot (for Linux, Mac OS X) is in your environmental variable **\$PATH**

# Plotting with Snap.py

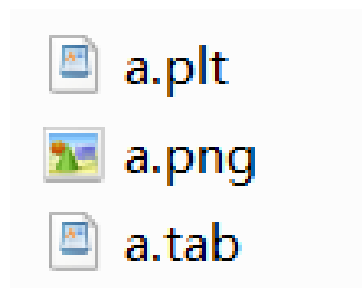
```
import snap
G = snap.LoadEdgeList(snap.PNGraph, "qa.txt", 1, 5)
snap.PlotInDegDistr(G, "Stack-Java", "Stack-Java In Degree")
```



Graph of Java QA on  
StackOverflow:  
in-degree distribution

# Snap.py + Gnuplot

- **Snap.py** generates three files:



- **.png** or **.eps** is the plot
- **.tab** file contains the data (tab separated file)
- **.plt** file contains the plotting commands

# Drawing Graphs

- **InstallGraphViz:**

<http://www.graphviz.org/>

- Make sure that the directory containing GraphViz is in your environmental variable **\$PATH**

# Drawing Graphs with Snap.py

```
G1 = snap.TNGraph.New()
```

Create graph

```
G1.AddNode(1)
```

```
G1.AddNode(5)
```

```
G1.AddNode(12)
```

```
G1.AddEdge(1, 5)
```

```
G1.AddEdge(5, 1)
```

```
G1.AddEdge(5, 12)
```



```
NIdName = snap.TIntStrH()
```

Set node labels

```
NIdName[1] = "1"
```

```
NIdName[5] = "5"
```

```
NIdName[12] = "12"
```

Draw

```
snap.DrawGViz(G1, snap.gvlDot, "G1.png", "G1", NIdName)
```

G1

# Snap.py Resources

- **Prebuilt packages** available for Mac OS X, Windows, Linux  
<http://snap.stanford.edu/snappy/index.html>
- **Snap.py documentation:**  
<http://snap.stanford.edu/snappy/doc/index.html>
  - Quick Introduction, Tutorial, Reference Manual
- **SNAP user mailing list**  
<http://groups.google.com/group/snap-discuss>
- **Developer resources**
  - Software available as open source under BSD license
  - GitHub repository  
<https://github.com/snap-stanford/snap-python>