**networkx.Graph.degree**

**Graph.degree**

A DegreeView for the Graph as G.degree or G.degree().

The node degree is the number of edges adjacent to the node. The weighted node degree is the sum of the edge weights for edges incident to that node.

This object provides an iterator for (node, degree) as well as lookup for the degree for a single node.

|  |  |
| --- | --- |
| **Parameters:** | * **nbunch** (*single node, container, or all nodes (default= all nodes)*) – The view will only report edges incident to these nodes. * **weight** (*string or None, optional (default=None)*) – The name of an edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node. |
| **Returns:** | * *If a single node is requested* * **deg** (*int*) – Degree of the node * *OR if multiple nodes are requested* * **nd\_view** (*A DegreeView object capable of iterating (node, degree) pairs*) |

**Examples**

>>>

**>>>** G = nx.path\_graph(4) *# or DiGraph, MultiGraph, MultiDiGraph, etc*

**>>>** G.degree[0] *# node 0 has degree 1*

1

**>>>** list(G.degree([0, 1, 2]))

[(0, 1), (1, 2), (2, 2)]

**networkx.Graph.neighbors**

**Graph.neighbors(*n*)**[**[source]**](https://networkx.github.io/documentation/stable/_modules/networkx/classes/graph.html#Graph.neighbors)

Return an iterator over all neighbors of node n.

This is identical to **iter(G[n])**

|  |  |
| --- | --- |
| **Parameters:** | **n** (*node*) – A node in the graph |
| **Returns:** | **neighbors** – An iterator over all neighbors of node n |
| **Return type:** | iterator |
| **Raises:** | [**NetworkXError**](https://networkx.github.io/documentation/stable/reference/exceptions.html#networkx.NetworkXError) – If the node n is not in the graph. |

**Examples**

>>>

**>>>** G = nx.path\_graph(4) *# or DiGraph, MultiGraph, MultiDiGraph, etc*

**>>>** [n **for** n **in** G.neighbors(0)]

[1]

**Notes**

It is usually more convenient (and faster) to access the adjacency dictionary as **G[n]**:

>>>

**>>>** G = nx.Graph() *# or DiGraph, MultiGraph, MultiDiGraph, etc*

**>>>** G.add\_edge('a', 'b', weight=7)

**>>>** G['a']

AtlasView({'b': {'weight': 7}})

**>>>** G = nx.path\_graph(4)

**>>>** [n **for** n **in** G[0]]

[1]

**networkx.algorithms.cluster.clustering**

**clustering(*G*, *nodes=None*, *weight=None*)**[**[source]**](https://networkx.github.io/documentation/stable/_modules/networkx/algorithms/cluster.html#clustering)

|  |  |
| --- | --- |
| **Parameters:** | * **G** (*graph*) * **nodes** (*container of nodes, optional (default=all nodes in G)*) – Compute clustering for nodes in this container. * **weight** (*string or None, optional (default=None)*) – The edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. |
| **Returns:** | **out** – Clustering coefficient at specified nodes |
| **Return type:** | [float](https://docs.python.org/2/library/functions.html#float), or dictionary |

**Examples**

>>>

**>>>** G=nx.complete\_graph(5)

**>>>** print(nx.clustering(G,0))

1.0

**>>>** print(nx.clustering(G))

{0: 1.0, 1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0}

**networkx.Graph.has\_node**

**Graph.has\_node(*n*)**[**[source]**](https://networkx.github.io/documentation/stable/_modules/networkx/classes/graph.html#Graph.has_node)

Return True if the graph contains the node n.

Identical to **n in G**

|  |  |
| --- | --- |
| **Parameters:** | **n** (*node*) |

**Examples**

>>>

**>>>** G = nx.path\_graph(3) *# or DiGraph, MultiGraph, MultiDiGraph, etc*

**>>>** G.has\_node(0)

True

It is more readable and simpler to use

>>>

**>>>** 0 **in** G

True

[Next](https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.Graph.__contains__.html)[Previous](https://networkx.github.io/documentation/stable/reference/classes/generated/networkx.Graph.__iter__.html)