# networkx

Networkx היא ספריית [קוד פתוח](https://github.com/networkx/networkx) בשפת פייתון ליצירת, לימוד וביצוע מניפולציות על גרפים.   
הספרייה מספקת:  
• כלים לחקר מבנה והדינמיקה של רשתות חברתיות, ביולוגיות ותשתיתיות.  
• ממשק תכנותי סטנדרטי ויישום של גרפים שמתאימים לאפליקציות רבות.  
• היכולת לעבודה במאמץ מינימאלי עם דאטה סטים לא סטנדרטים.   
היות והספרייה ענקית נמליץ מראש להסתכל על [האתר הרשמי](https://networkx.org/) של הספרייה לעוד פרטים.

התקנה:

pip install networkx

פעולות בסיסיות בגרפים-   
networx מספקת כלי לשימוש בdata farme לא סטנדרטי כל כך- בפנדס היה לנו מבנה רלאציוני שנשען על האובייקט מערך של numpy, networkx לעומת זאת משתמשת במבנה של גרף לשמירה על הנתונים, וכגרף ניתן לבצע עליו מניפולציות מתוך תורת הגרפים למשל מציאת שידוך מקסימלי, מציאת מסלול קצר, קוטר צביעה ועוד.  
  
תחילה נראה כיצד לייצר גרף, להוסיף לו קשתות וצמתים.  
יצירת גרף לא מכוון תעשה דרך שימוש במחלקה Graph של הספרייה, או במחלקה DiGraph ליצירת גרף מכוון, אפשר לאתחל את הגרף מראש עם גרף אחר, או עם רשימה של צלעות.  
בשביל להוסיף צמתים לגרף נשתמש בכמה פונקציות, הראשונה היא המתודה add\_node() שמקבלת את שם הצומת ושומרת אותו כקודקוד בודד.  
הדרך השנייה היא להוסיף ישירות שני קודקודים, עם המתודה add\_nodes\_from() המתודה מקבלת רשימה של כמה קודקודים ומכניסה אותם לגרף, ואפשר גם להוסיף תכונות לקודקוד אם מוסיפים לו בנוסף לערך גם מילון עם התכונות של אותו קודקוד:

import networkx as nx  
  
my\_graph = nx.Graph()

my\_graph.add\_node(1)

my\_graph.add\_nodes\_from([2, 3])

אפשר גם ליצור גרף רנדומלי עם הפונקציה path\_graph() ולספק לה מספר ממנו היא תיקח את ערכיה.   
וכמו כן אפשר גם להכניס גרף אחד באחר(להכניס את הצמתים שלו אחד אחד) עם המתודה add\_node\_from() , או להכניס גרף אחר כצומת אחד בגרף עם מתודהadd\_node() , למה שנרצה לעשות את זה? תחשבו על האינטרנט, היא רשת של רשתות, יש לה רשת ברמת המקרו ורשת פנימית ברמת המיקרו:

graph\_other = nx.path\_graph(10)

my\_graph.add\_nodes\_from(graph\_other)

my\_graph.add\_node(graph\_other)

הוספת ענפים בגרף נעשה עם המתודה add\_edge() ונכניס בה את הצמתים שבניהם קיימת קשת.  
אפשר גם להכניס כמה קשתות כאשר כל קשת מצוינת ב-tuple עם ערכי שני קוקודים והתכונה שאותה נושאת הקשת(למשל משקל) במילון, כל גרף שומר את הקשתות שלו במתנה בשם edges:

my\_graph.add\_edge(1, 2)

e = (2, 3)

my\_graph.add\_edge(\*e)

my\_graph.add\_edges\_from([(1, 2), (1, 3),(2, 3, {'weight': 3.1415})])  
  
list(my\_graph.edges)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
[(1, 2), (1, 3), (2, 3)]

my\_graph.clear()

בשביל לנקות את הגרף נשתמש בפונקציה clear().   
  
  
  
לכל הגרפים יש כמה שדות: צמתים, קשתות, טבלאת שכנים, טבלת דרגות בגרף.  
יש גם את הפונקצייה successor שמאפשרת לראות אילו קודקודים מחוברים לקודקוד ספציפי בקשת.

G = nx.DiGraph()

G.add\_edge(2, 1) # adds the nodes in order 2, 1

G.add\_edge(1, 3)

G.add\_edge(2, 4)

G.add\_edge(1, 2)

G.add\_edge(3, 2)

assert list(G.successors(2)) == [1, 4]

assert list(G.edges) == [(2, 1), (2, 4), (1, 3), (1, 2),(3, 2)]

print(list(G.nodes))

print(list(G.edges))

print(list(G.adj[1])) # or list(G.neighbors(1))

print(G.degree[1]) # the number of edges incident to 1  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
[2, 1, 3, 4]

[(2, 1), (2, 4), (1, 3), (1, 2), (3, 2)]

[3, 2]

3

הסרה של צמתים או קשתות יעשו עם המתודות remove\_node() להוצאת קדקוד בודד כשהארגומנט הוא שם הקדקוד,  
 הוצאת כל הקדקודים שמחוברים לקדקוד ספציפי עם המתודה remove\_nodes\_from().  
הוצאת צלע ספציפית עם המתודה remove\_edge() והארגומנט צריך להיות הקדקודים שיש בניהם צלע.   
והוצאת כמה צלעות עם המתודה remove\_edges\_from() והארגומנט שלה הוא רשימה של צלעות שצריך להוציא מהגרף.

G.add\_edge(4,1)

G.add\_edge(4,5)

G.add\_edge(5,1)

G.add\_edge(2,5)

G.add\_edge(6,5)

G.add\_edge(1,6)

G.add\_edge(3,5)

G.add\_edge(4,6)

G.remove\_node(3)

G.remove\_nodes\_from('4')

G.remove\_nodes\_from('5')

G.remove\_edge(6, 5)

https://networkx.org/documentation/latest/tutorial.html#directed-graphs