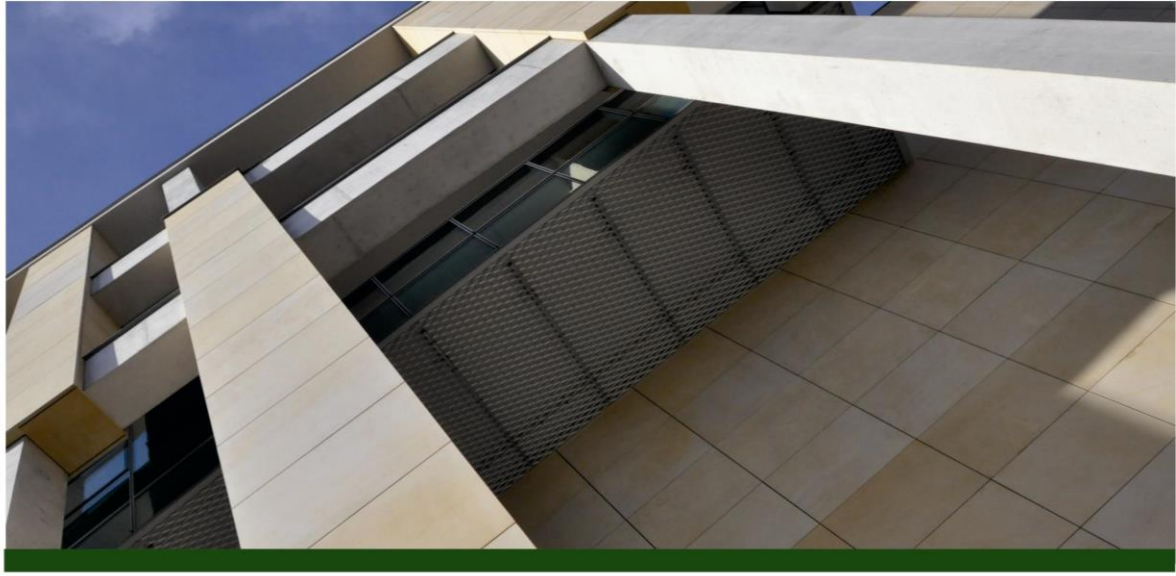


Data Science

Analiza sieci społecznych w Pythonie

Jupyter

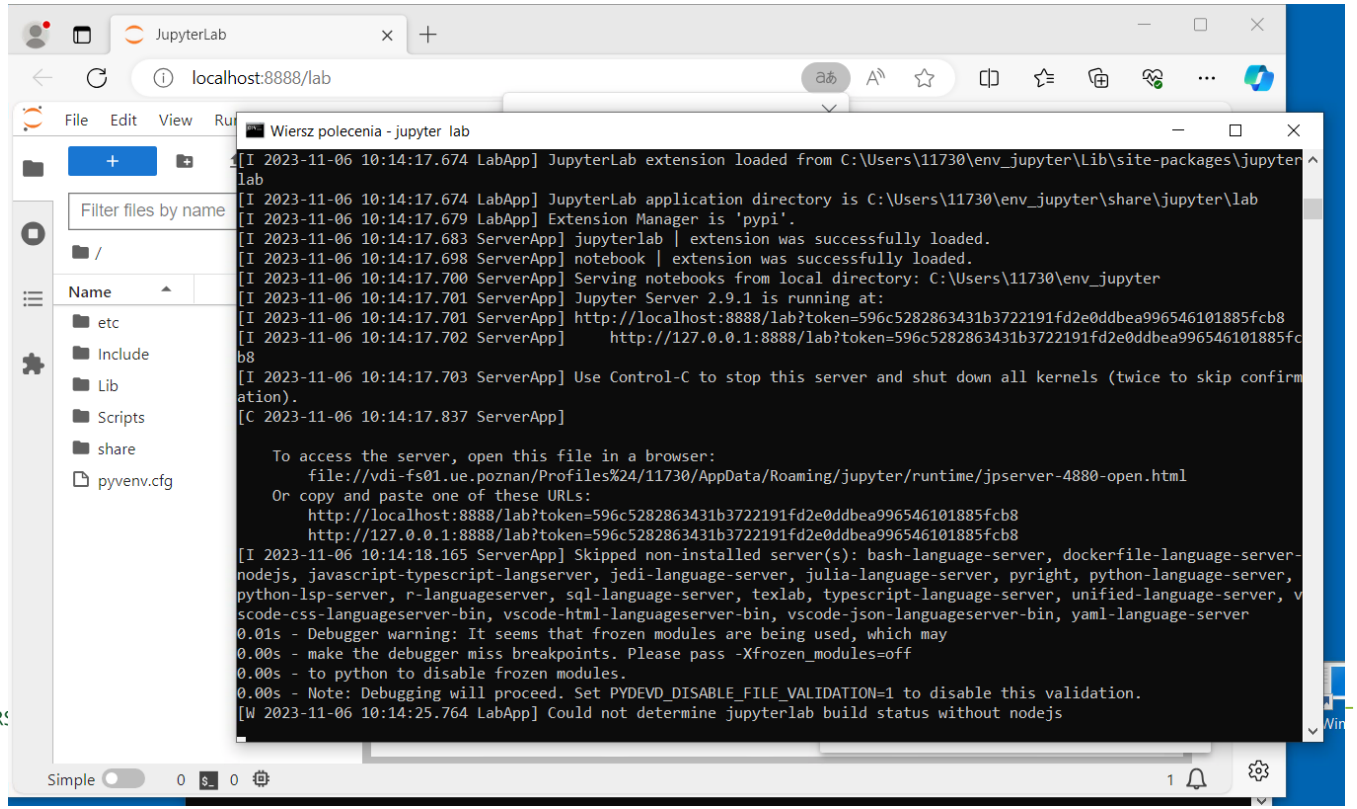
Jupyter

- interaktywne środowisko do pracy z kodem w Pythonie w przeglądarce WWW
- zawiera zarówno kod, wraz z wynikami jego wykonania, jak i elementy opisowe (tekst, rysunki, LaTeX)
- wywołanie Jupyter
  - uruchamiamy linię poleceń (cmd)
    - przechodzimy do właściwego katalogu (cd): c:\users\{NIU}
  - tworzymy środowisko: (python –m venv env_jupyter)
  - aktywujemy środowisko: (env_jupyter\Scripts\activate.bat)
  - instalujemy potrzebne moduły (pip install jupyter networkx)
  - wywołujemy program (jupyter lab)
  - program otwiera się w przeglądarce

Wyniki po instalacji/starcie

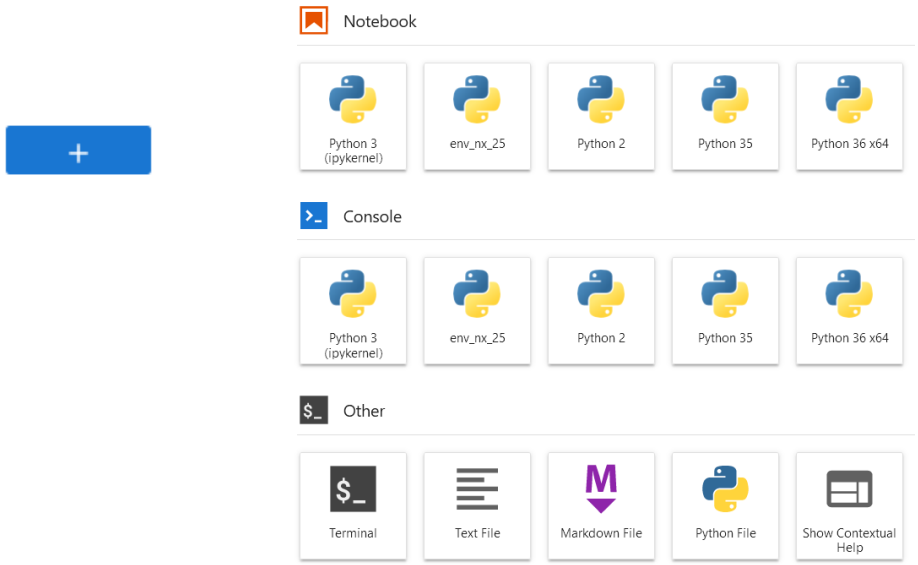
```
(env_jupyter) C:\Users\11730\env_jupyter>pip install networkx
Collecting networkx
  Obtaining dependency information for networkx from https://files.pythonhosted.org/packages/d5/f0/8fbc882ca80cf077f1b24
6c0e3c3465f7f415439bdea6b899f6b19f61f70/networkx-3.2.1-py3-none-any.whl.metadata
  Downloading networkx-3.2.1-py3-none-any.whl.metadata (5.2 kB)
Downloading networkx-3.2.1-py3-none-any.whl (1.6 MB)
   ---------------------------------------- 1.6/1.6 MB 4.2 MB/s eta 0:00:00
Installing collected packages: networkx
Successfully installed networkx-3.2.1

[notice] A new release of pip is available: 23.2.1 -> 23.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```



Nowy notatnik

- tworzymy nowy notatnik poprzez niebieski plus



- omówienie filozofii pracy
  - komórki z kodem
  - wynik działania
  - komórki z tekstem

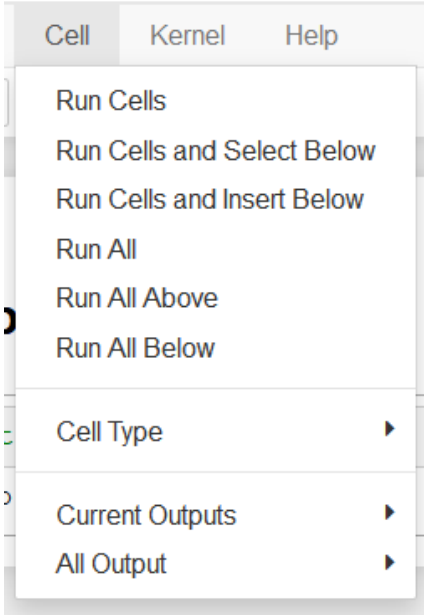
Pierwszy program

- kernel – silnik wykonawczy, odpowiada za uruchamianie kodu w określonym języku
- zachowuje w pamięci bieżące wartości zmiennych
- wpisujemy w komórkę kodu odpowiednią treść programu

```
In [1]: print("Hello world!")

        Hello world!
```

- wykonanie kodu (Ctrl+Enter)

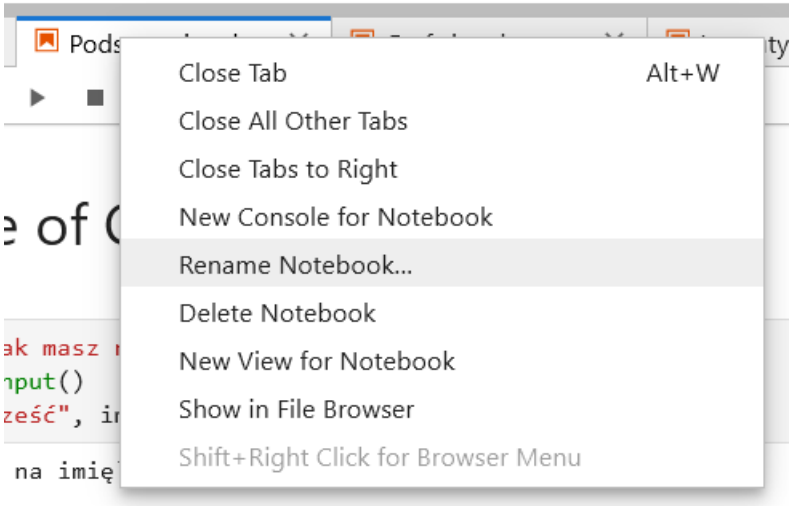


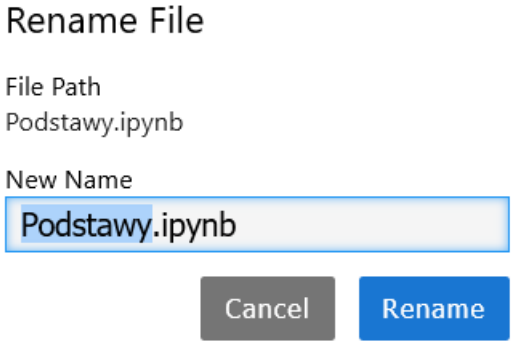
Importowanie bibliotek

- analogicznie do zwykłego programu w Pythonie
- biblioteki mogą być zaimportowane w dowolnym momencie

Zapisywanie pliku

- domyślna nazwa to „Untitled”
- wybieramy menu File->Rename notebook, aby zmienić





- „Save and checkpoint” powoduje zapisanie na dysku
- plik *Hello.ipynb* zapisany jest w folderze, z którego uruchomiliśmy Jupyter

Skróty klawiszowe

- dostępne komendy po poleceniu Ctrl+Shift+p
- A – wstawia komórkę ponad (above)
- B – wstawia komórkę pod (below)
- M – zmiana na tekst (markdown)
- Y – zmiana na kod
- Ctrl+Enter – wykonanie komórki
- Shift+Enter – wykonanie i przejście do następnej
- Alt+Enter – wykonanie i dodanie nowej poniżej

- opis języka Markdown
  - https://www.markdownguide.org/

Podstawy języka Python

- (zakładam, że język jako taki pojawił się na zajęciach)

- możliwość interakcji z użytkownikiem

```
In [1]: print("Jak masz na imię?")
        imie = input()
        print("Cześć ", imie)

        Jak masz na imię?
        John
        Cześć  John
```

Podstawy języka Python c.d.

- warunki logiczne

```
In [3]: if 5 > 0:
            print("Pięć jest większe od zera")
        print("Koniec programu")

Pięć jest większe od zera
Koniec programu
```

- warunki wielokrotne

```
In [5]: x = 5
        if x<2:
            print("Mało")
        elif x<5:
            print("Średnio")
        else:
             print("Dużo")

Dużo
```

Podstawy języka Python c.d.

- pętle (liczba powtórzeń)

```
In [6]: for i in range(6):
            print(i)
```

```
0
1
2
3
4
5
```

- iteracja po elementach tablicy

```
In [8]: tablica = ["Alicja", "Bob", "Celina"]
        for t in tablica:
            print("Hello, {}!".format(t))
```

```
Hello, Alicja!
Hello, Bob!
Hello, Celina!
```

Podstawy języka Python c.d.

- indeksowanie kolekcji
  - krotki

```
In [11]: collection = ("Dania","Belgia","Finlandia")
         collection[0]

Out[11]: 'Dania'

In [12]: collection[-2]

Out[12]: 'Belgia'
```

  - słowa

```
In [13]: word = 'Uniwersytet'
         word[2], word[-1]

Out[13]: ('i', 't')
```

  - tablice

```
In [14]: list = ['katedra','uniwersytet','instytut']
         list[1], list[-3]

Out[14]: ('uniwersytet', 'katedra')
```

Podstawy języka Python c.d.

- wycinki z kolekcji

```
In [15]: collection = ("Dania","Belgia","Finlandia")
         collection[0:2]
Out[15]: ('Dania', 'Belgia')

In [18]: collection[-2:]
Out[18]: ('Belgia', 'Finlandia')

In [16]: word = 'Uniwersytet'
         word[0:3]
Out[16]: 'Uni'

In [19]: word[-5:-1]
Out[19]: 'syte'

In [17]: list = ['katedra','uniwersytet','instytut']
         list[:1]
Out[17]: ['katedra']

In [20]: list[-2:]
Out[20]: ['uniwersytet', 'instytut']
```

Podstawy języka Python c.d.

- tradycyjne podejście do obliczeń na zbiorze elementów

```
[48]: w = range(6)
      w

[48]: range(0, 6)

[49]: x = []
      for i in w:
          ii = i*i
          x.append(ii)
      x

[49]: [0, 1, 4, 9, 16, 25]
```

- Python proponuje znaczne uproszczenia, umożliwiające również optymalizację wykonania

Podstawy języka Python c.d.

- wyrażenia tablicowe

```
[50]: y = [i for i in w]
      y

[50]: [0, 1, 2, 3, 4, 5]

[51]: y = [i*i for i in w]
      y

[51]: [0, 1, 4, 9, 16, 25]
```

- map – wykonanie funkcji na tablicy (funkcja via lambda)
  - komentarz nt. „lazy evaluation” w Pythonie

```
[52]: z = map(lambda i: i*i, w)
      z

[52]: <map at 0x256b7e67f40>

[53]: list(z)

[53]: [0, 1, 4, 9, 16, 25]
```

Grafy i sieci społeczne

Grafy w Pythonie

- biblioteka NetworkX
- http://networkx.github.io/
- *NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.*
- struktury: grafy, grafy skierowanie, wielografy
- generatory grafów, import, eksport
- wiele standardowych algorytmów
- tutorial: https://github.com/jtorrents/pydata_bcn_NetworkX

Wprowadzenie do grafów

- notebook Grafy.ipynb
- dołączenie odpowiednich bibliotek (+aliasy)

```
[1]: import networkx as nx
     import matplotlib.pyplot as plt
     %matplotlib inline
     print('NetworkX version: {}'.format(nx.__version__))

     NetworkX version: 3.2.1
```

- utworzenie grafu nieskierowanego

```
In [4]: G = nx.Graph()
```

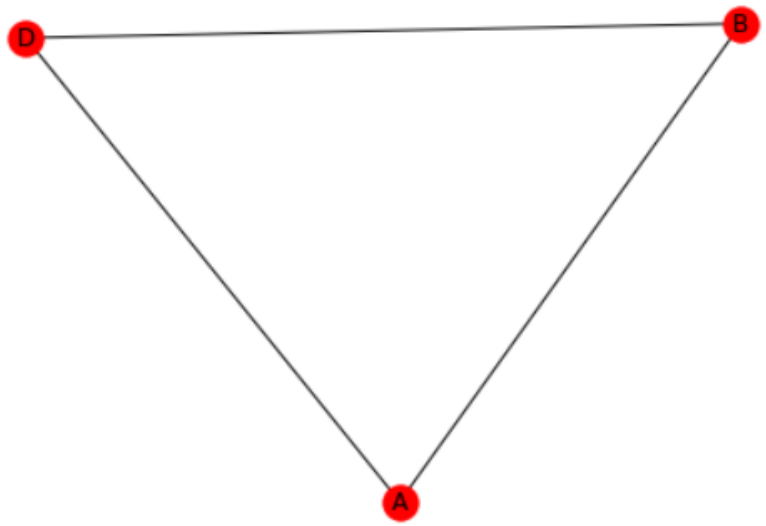
- utworzenie grafu skierowanego

```
In [5]: D = nx.DiGraph()
```

Wprowadzenie do grafów c.d.

- dodanie krawędzi i wykreślenie grafu

```
In [11]: G.add_edges_from([('A', 'B'), ('A', 'D'), ('B', 'D')])
         nx.draw(G, with_labels=True)
```



Wprowadzenie do grafów c.d.

- sposób reprezentacji

Ze względu na sposób reprezentacji grafu możliwe są działania:

- **n in G**, aby sprawdzić, czy graf G zawiera węzeł n
- **for n in G**, aby iterować po wszystkich węzłach
- **G[n]**, aby uzyskać dostęp do wszystkich sąsiadów n w G
- **len(G)**, aby uzyskać liczbę węzłów w G

- lista sąsiedztwa

```
In [14]: print(G.adj)
{'A': {'B': {}, 'D': {}}, 'B': {'A': {}, 'D': {}}, 'D': {'A': {}, 'B': {}}}
```

Wprowadzenie do grafów c.d.

- pozostałe operacje

```
In [15]: 'A' in G

Out[15]: True

In [17]: for n in G:
             print(n)

A
B
D

In [18]: print(G['A'])

{'B': {}, 'D': {}}

In [19]: len(G)

Out[19]: 3
```

Wprowadzenie do grafów c.d.

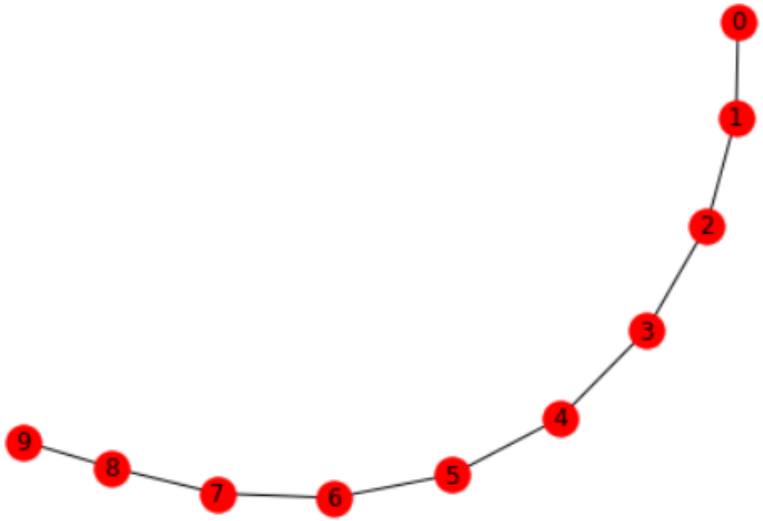
- dodawanie węzła
  - można dodać węzeł, ale będzie on odłączony od innych węzłów

```
In [20]: G.add_node('Z')
         nx.draw(G, with_labels=True)
```

Wprowadzenie do grafów c.d.

- wbudowane grafy

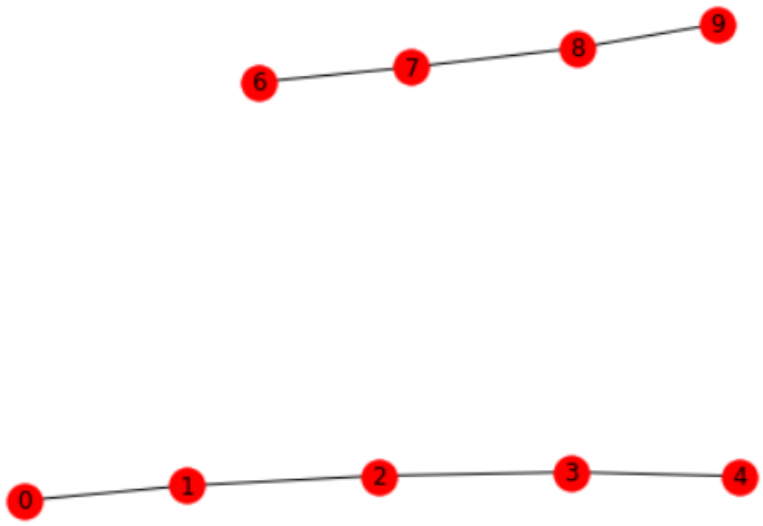
```
In [21]: H = nx.path_graph(10)
         nx.draw(H, with_labels=True)
```



Wprowadzenie do grafów c.d.

- usunięcie węzła

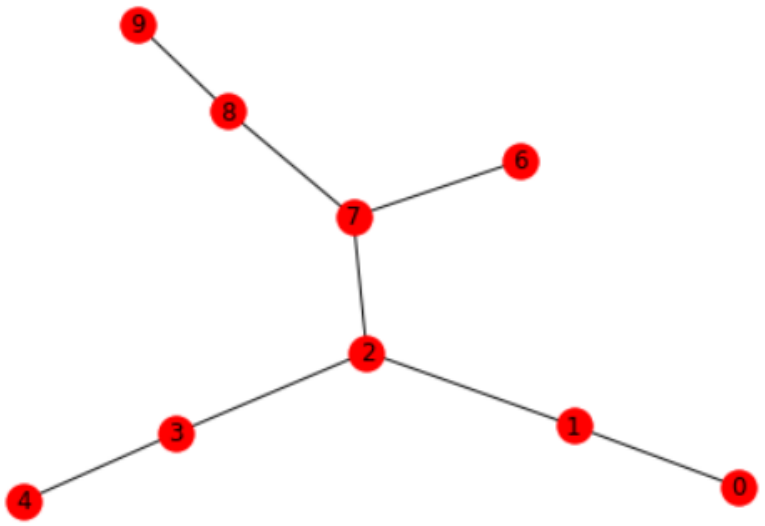
```
In [24]: H.remove_node(5)
         nx.draw(H, with_labels=True)
```



Wprowadzenie do grafów c.d.

- dodanie nowej krawędzi

```
In [25]: H.add_edge(2,7)
         nx.draw(H, with_labels=True)
```

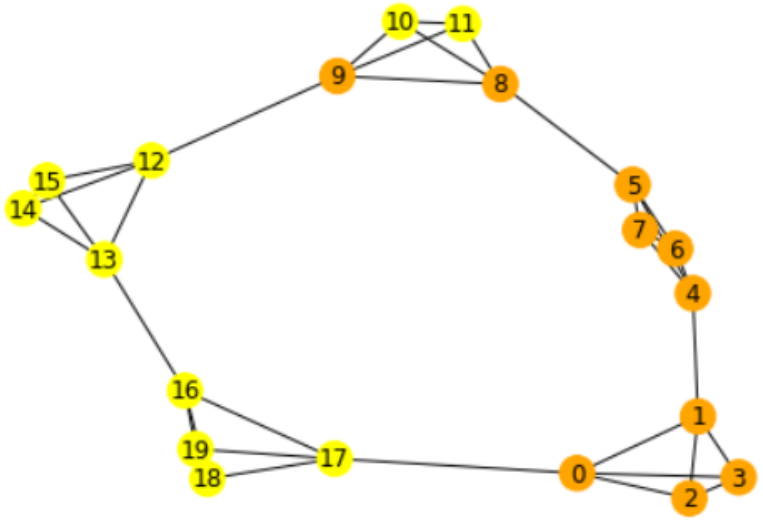


Wizualizacja grafów

Wprowadzenie do grafów c.d.

- kolorowanie węzłów (poprzez color map)

In [44]:
```
L = nx.ring_of_cliques(5, 4)
color_map = []
for node in L:
    if int(node) <10:
        color_map.append('orange')
    else: color_map.append('yellow')
nx.draw(L,node_color = color_map,with_labels = True)
```



Rozkład grafów

- Dostępne różne algorytmy
  - wbudowane, np. spring
  - dostępne poprzez graphviz
  - zewnętrzne biblioteki, np. force atlas
- przykładowe notatniki:
  - Layouty.ipynb
  - Force atlas.ipynb

Przykładowy graf

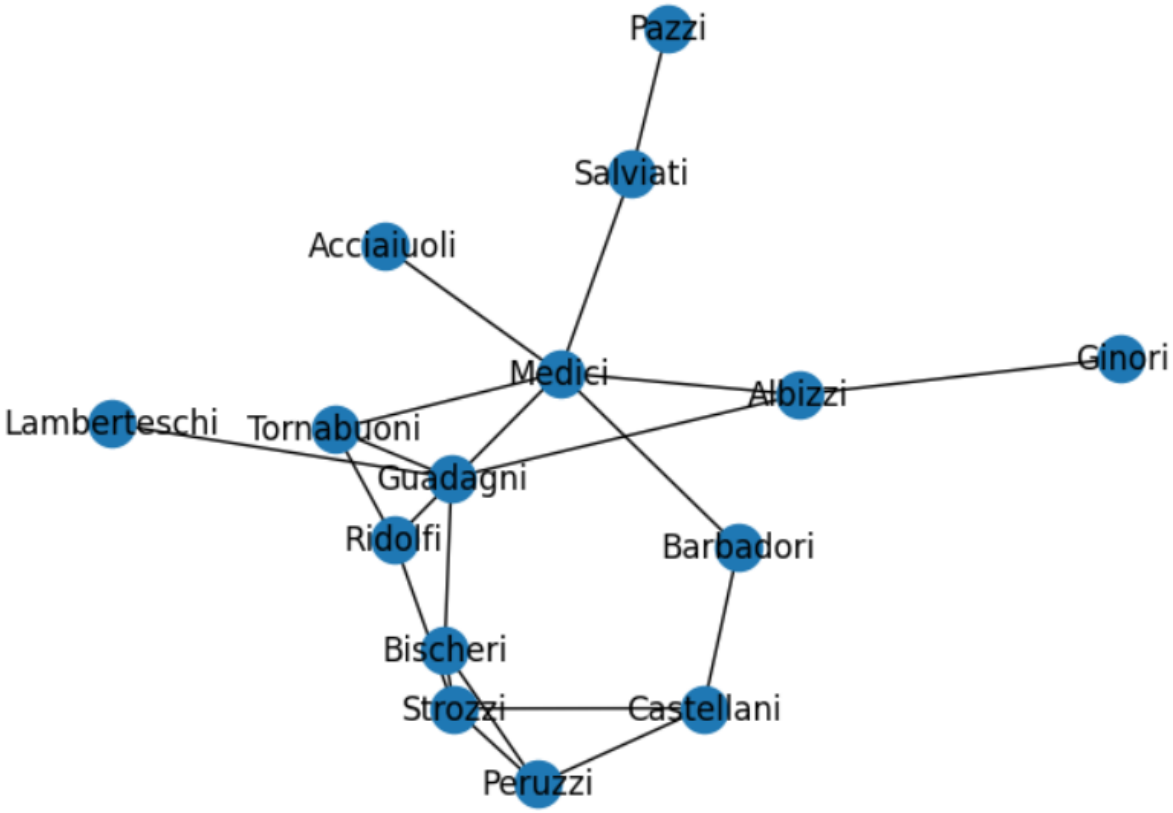
- przykład rozkładów dla rodzin florenckich

```
[3]: G = nx.florentine_families_graph()
     print(G)

     Graph with 15 nodes and 20 edges
```

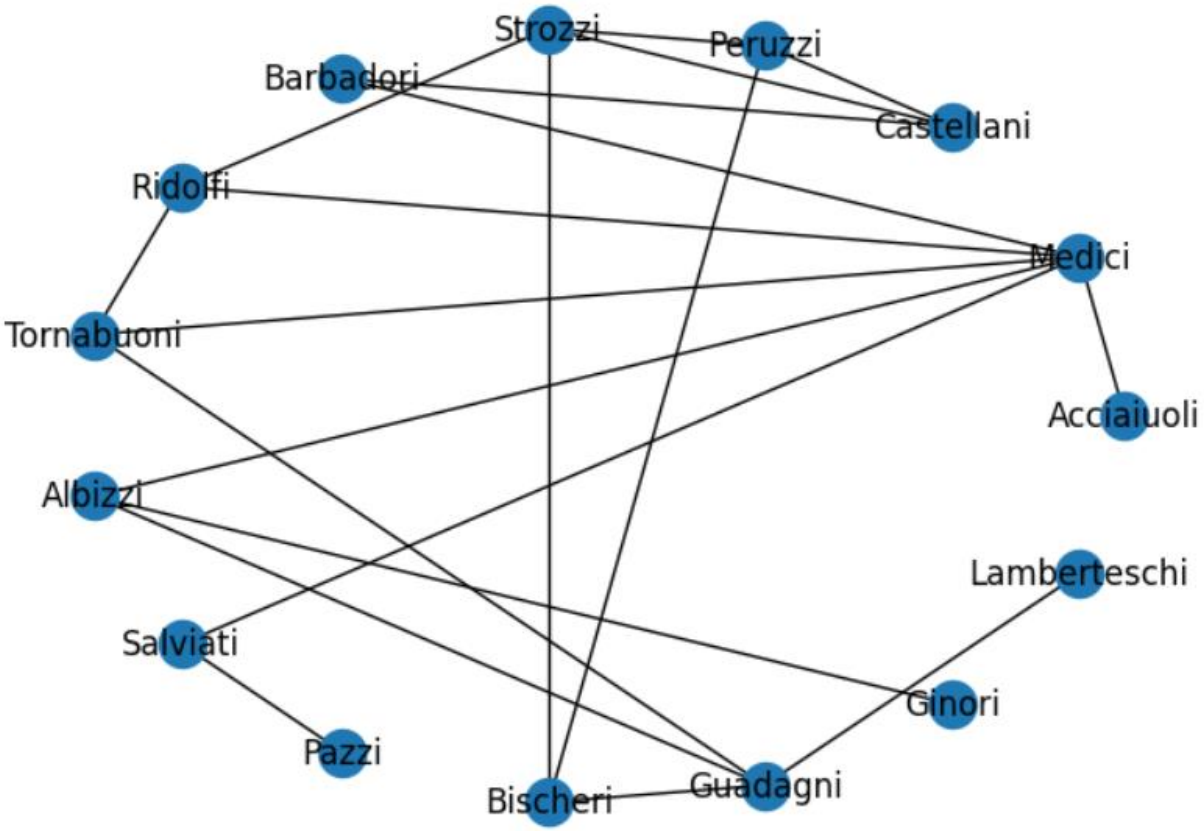
Rozkład spring

```
[5]: pos_spring = nx.spring_layout(G)
     nx.draw(G, pos_spring, with_labels=True)
```



Rozkład kołowy

```
[8]: pos_cir = nx.circular_layout(G)
     nx.draw(G, pos_cir, with_labels=True)
```



Centralność

Centralność

- różne miary, które określają ważność węzła lub krawędzi
- ważność może wynikać
  - z środkowego położenia
  - ze spajania różnych społeczności
  - bycia w wielu najkrótszych ścieżkach
- miary niekoniecznie są jakoś ze sobą skorelowane, co pokazuje późniejsza analiza
- dokumentacja: https://networkx.github.io/documentation/stable/reference/algorithms/centrality.html

Degree centrality

- stopień (degree) – liczba podłączonych węzłów do danego węzła
- degree centrality – liczba powiązanych węzłów przez liczbę wszystkich potencjalnych sąsiadów
- można to potraktować jako znormalizowane degree

Betweenness centrality

- wyznaczane są najkrótsze trasy między każdą parą węzłów
- betweenness centrality dla węzła to suma odsetków wszystkich najkrótszych tras, które przechodzą przez ten węzeł

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t)-paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s,t. If $s = t$, $\sigma(s,t) = 1$, and if $v \in s,t$, $\sigma(s,t|v) = 0$ [2].

Closeness centrality

- closeness centrality danego węzła to odwrotność średniej najkrótszej ścieżki z pozostałych węzłów do danego węzła
- czyli potocznie jak dany węzeł „ma daleko” do innych

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)},$$

where `d(v, u)` is the shortest-path distance between `v` and `u`, and `n` is the number of nodes that can reach `u`.

Eigenvector centrality

- odniesienie do wektorów i wartości własnych macierzy
- eigenvector centrality przypisuje wartości względne
  - tym większe, im więcej powiązań z węzłami o wysokiej wartości tej miary
  - tym mniejsze, im więcej powiązań z węzłami o małej wartości tej miary
- wyliczenie następuje iteracyjnie
- skojarzenia z PageRank są jak najbardziej słuszne

Wyliczenie degree centrality

- przykład w notebooku: Centrality.ipynb

In [4]:
```
from operator import itemgetter
degc = nx.degree_centrality(G)
# let's list the scores
sorted(degc.items(), key=itemgetter(1), reverse=True)
```

Out[4]:
```
[('Medici', 0.42857142857142855),
 ('Strozzi', 0.2857142857142857),
 ('Guadagni', 0.2857142857142857),
 ('Castellani', 0.21428571428571427),
 ('Bischeri', 0.21428571428571427),
 ('Ridolfi', 0.21428571428571427),
 ('Albizzi', 0.21428571428571427),
 ('Tornabuoni', 0.21428571428571427),
 ('Peruzzi', 0.21428571428571427),
 ('Salviati', 0.14285714285714285),
 ('Barbadori', 0.14285714285714285),
 ('Ginori', 0.07142857142857142),
 ('Pazzi', 0.07142857142857142),
 ('Lamberteschi', 0.07142857142857142),
 ('Acciaiuoli', 0.07142857142857142)]
```

Odwzorowanie wartości na kolory

- najpierw tworzymy tablicę z wartościami dla poszczególnych węzłów. Uwaga: degc jest słownikiem.

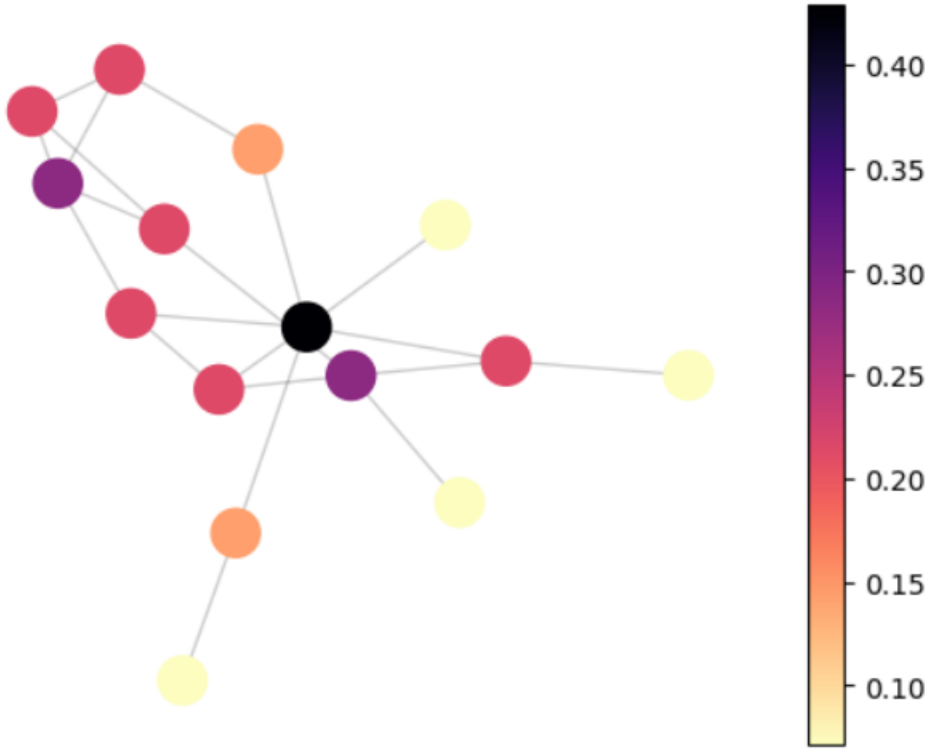
```
In [13]: nodes = G.nodes()
         values = [degc.get(node) for node in nodes]
         values

Out[13]: [0.21428571428571427,
          0.07142857142857142,
          0.14285714285714285,
          0.21428571428571427,
          0.2857142857142857,
          0.42857142857142855,
          0.2857142857142857,
          0.07142857142857142,
          0.14285714285714285,
          0.21428571428571427,
          0.21428571428571427,
          0.07142857142857142,
          0.21428571428571427,
          0.07142857142857142,
          0.21428571428571427]
```

- wartości przekładamy na kolory za pomocą colormap
  - atrybuty: node_color=, cmap=

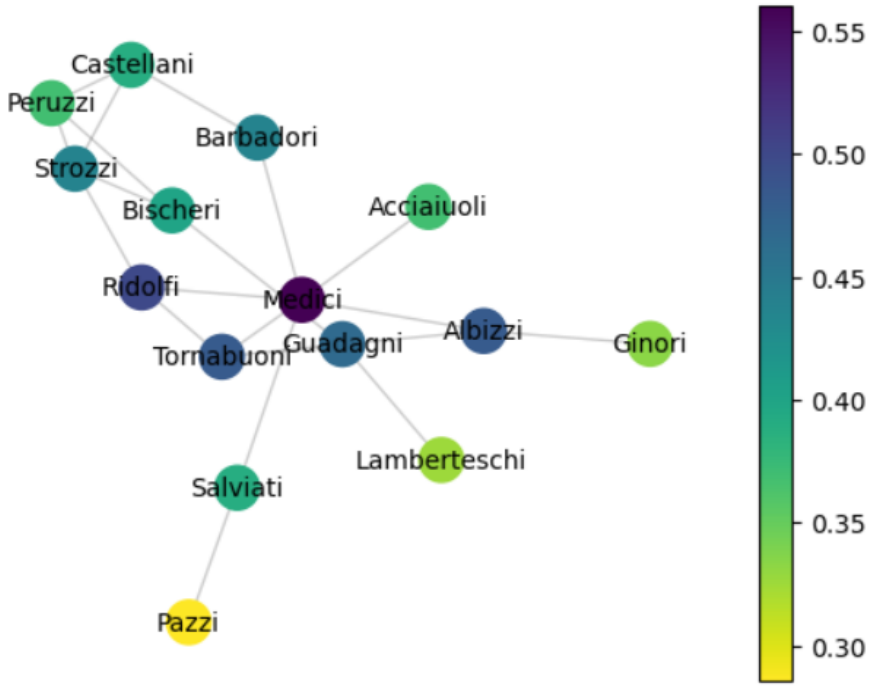
Wykres degree centrality

[9]:
```
ec = nx.draw_networkx_edges(G, pos_spring, alpha=0.2)
nc = nx.draw_networkx_nodes(G, pos_spring, nodelist=nodes, node_color=values,
                            cmap=plt.cm.magma_r)
plt.colorbar(nc)
plt.axis('off')
plt.show()
```



Wykres betweenness centrality

[28]:
```
values = [clos.get(node) for node in nodes]
labels = {x:x for x in G.nodes()}
ec = nx.draw_networkx_edges(G, pos_spring, alpha=0.2)
nc = nx.draw_networkx_nodes(G, pos_spring, nodelist=nodes, node_color=values,
                            cmap=plt.cm.viridis_r)
nx.draw_networkx_labels(G, pos_spring, labels, font_size=10, font_color="black")
plt.colorbar(nc)
plt.axis('off')
plt.show()
```



Wykrywanie społeczności

- notebook: Community detection.ipynb
- gł. metoda Louvain Modularity
  - służy do wykrywania społeczności w dużych sieciach
  - optymalizacja zachłanna O(n log n)
  - modularity – wartość z przedziału -1 do 1, mierzy gęstość węzłów wewnątrz społeczności do węzłów poza społecznością
  - algorytm maksymalizuje wartość modularity, odpowiednio dobierając społeczności
- naturalnie sieci, np. mózg, posiadają wysokie modularity

Wizualizacja przykładowej społeczności

[11]:
```
from community import community_louvain
```

[12]:
```
parts = community_louvain.best_partition(G_fb)
values = [parts.get(node) for node in G_fb.nodes()]
```

[13]:
```
plt.axis("off")
nx.draw_networkx(G_fb, pos = spring_pos, cmap = plt.get_cmap("jet"), node_color = values,
                 node_size = 35, with_labels = False)
```