ELEMENTARNE WPROWADZENIE DO

PYTHONA

ELEMENTY TEORII GRAFÓW i ALGORYTMY GRAFOWE

materiały dla kierunku nauczanie matematyki i informatyki

ale nie tylko

Katarzyna Rybarczyk-Krzywdzińska

Spis treści

Python 3.7

TYPY OBIEKTÓW DLA ALGORYTMÓW GRAFOWYCH (skrót):

W przykładach będziemy wykorzystywać graf G=(V,E) z wagami zadanymi funkcją w() o

- zbiorze wierzchołków V={a,b,c,d,e};
- zbiorze krawędzi E={ab,ad,bc,bd,cd,ce}
- macierzy wag:

∞	1.2	∞	10	∞
1.2	∞	1	8	∞
∞	1	∞	7	12
10	8	7	∞	∞
∞	∞	12	∞	∞

Jego macierz wag można zapisać w pliku tekstowym matrix.txt w ten sposób (- oznacza ∞):

```
- 1.2 – 10 -
1.2 – 1 8 -
- 1 – 7 12
10 8 7 - -
- - 12 - -
```

Kolejne wczytywane linie matrix.txt są czytane jako **ŁAŃCUCHY** np. pierwsza linia:

```
'- 1.2 – 10 -'
```

ŁAŃCUCHY można rozbić na **LISTĘ** łańcuchów:

```
['-','1.2','-','10','-']
```

Natomiast łańcuchy z **LISTY** można zamienić na **LICZBY** (float('inf') działa jak ∞):

```
[float('inf'),1.2,float('inf'),10,float('inf')]
```

Wierzchołki, krawędzie i wagi można zapisać jako **LISTY** (łańcuchów, list, liczb):

```
wierzch=['a','b','c','d','e']
kraw=[['a','b'],['a','d'],['b','c'],['b','d'],['c','d'],['c','e']]
wagi=[1.2,10,1,8,7,12]
```

Można je też zapisać jako, analogiczne do list, KROTKI (krotek nie można modyfikować) np.

```
krawplus=(('a','b'),('a','d'),('b','c'),('b','d'),('c','d'),('c','e'))
wagiplus=(1.2,10,1,8,7,12)
```

Macierz wag można zapisać jako **LISTĘ LIST** (tutaj inf=float('inf')):

```
matrix=[[inf,1.2,inf,10,inf],[1.2,inf,1,8,inf],[inf,1,inf,7,12],[10,8,7,inf,inf]],
[inf,inf,12,inf,inf]]
```

SŁOWNIKI wygodnie się wykorzystuje do zapisu listy następników, listy wag krawędzi, całego grafu:

```
nast={'a':['b','d'],'b':['a','c','d'],'c':['b','d','e'],'d':['a','b','c'],'e':['c']}
wagikraw={('a','b'):1.2,('a','d'):10,('b','c'):1,('b','d'):8,('c','d'):7,('c','e'):12}
graf={'a':{'b':1.2,'d':10},'b':{'a':1.2,'c':1,'d':8},'c':{'b':1,'d':7,'e':12},
        'd':{'a':10,'b':8,'c':7},'e':{'c':12}}
```

Wykorzystanie **ZBIORÓW**:

```
zbwierzch={'a','b','c','d','e'}
zbkraw={('a','b'),('a','d'),('b','c'),('b','d'),('c','d'),('c','e')}
```

WAŻNE: w Pythonie istotne są „wcięcia”.

INSTRUKCJE WARUNKOWE

if *tutaj zapisany warunek, który ma być spełniony* **:**

    tutaj lista komend, które mają być wykonane, jeśli warunek spełniony

elif *kolejny warunek, sprawdzany, jeśli poprzedni nie jest spełniony* **:**

    tutaj lista komend, które mają być wykonane, jeśli drugi warunek spełniony

else:

    tutaj lista komend do wykonania, jeśli żaden warunek nie jest spełniony

UWAGA1: elif i else są opcjonalne, elif można zastosować kilka razy pod rząd.

UWAGA2: Nie zapomnij o ':' i o wcięciach!!!

Przykłady warunków w instrukcji warunkowej if:

if a == b: (jeśli a równe b)

if a != b: (jeśli a różne od b)

if a<b:

if a>b:

if a>=b:

if a<=b:

PĘTLE

(UWAGA: nie zapomnij o „:” i wcięciach)

while *sprawdzany warunek* **:**

tutaj lista komend, które mają być wykonane, jeśli warunek spełniony

for z **in** *tutaj obiekt (łańcuch,lista,słownik) z którego pobieramy kolejne z* **:**

tutaj lista komend, które mają być wykonane, na kolejnych z

wersja pętli for, gdy chcemy powtórzyć pętlę n razy (np. 100 razy)

for i **in range(**n**):**

tutaj lista komend, które mają być wykonane n razy

Powyżej w kolejnych iteracjach „i” przyjmuje kolejno wartości 0,1,…,n-1.

Gdy chcemy powtórzyć dla liczb „i” z zakresu od a do b (uwaga na b+1!!!):

for i **in range(**a,b+1**):**

tutaj lista komend, które mają być wykonane dla i=a,…,b

lub

for i **in range(**a,b,3**):**

tutaj lista komend, które mają być wykonane dla i=a,a+3,a+6,…,x,

gdzie x jest największą liczbą całkowitą postaci a+3k ***mniejszą*** *od b*

break (Wychodzi z najbliższej obejmującej pętli, omija całą instrukcję tej pętli)

continue (Przechodzi do nagłówka najbliższej obejmującej pętli)

ISTOTNE TYPY OBIEKTÓW/ZMIENNYCH w PYTHONIE (szczegóły):

UWAGA: Podając wynik działania komendy, zakładamy, że jej działanie dotyczy pierwotnego obiektu zdefiniowanego na początku rozdziału (nie uwzględniamy działania poprzednio przedstawionych komend).

ZMIENNE LOGICZNE

przyjmują wartości True lub False

LICZBY

Nas będą interesować dwa typy **int** (l. całkowite) i **float** (l. zmiennoprzecinkowe). Python automatycznie przypisuje typ.

- lwierzch=100 (typ int)
- lkraw=200 (typ int)
- waga = 1.876 (typ float)
- inf=**float('inf')** (tak można przypisać zmiennej inf wartość nieskończoność – typ float)

Komendy, które na pewno się przydadzą:

- **int**(waga) (Zamienia float na int i daje część całkowitą z float. Tu da 1)
- **float**(lkraw) (Zamienia int na float)
- +,-,*,/ ,**,%,//(dadawanie, odejmowanie, mnożenie, dzielenie, potęgowanie, reszta z dzielenia, część całkowita z dzielenia. UWAGA: wynik dzielenia jest zawsze float)
- szybkie dodawanie, mnożenie itp:
  - waga+=2 (przypisuje zmiennej waga wartość waga+2. Tutaj 3.876)

Komendy, które może się przydadzą

- **round**(waga, 2) (Zaokrągla do zadanej liczby miejsc po przecinku (tutaj do dwóch). Tutaj da 1,88)

ŁAŃCUCHY

- napis='moj graf' (zmiennej napis przypisaliśmy łańcuch 'moj graf')
- linia='- 1.2 – 10 -'
- waga='37'

Komendy, które na pewno się przydadzą:

- linia.**split**() (Podzieli na części zgodnie z ,,pustymi" znakami. Zwróci listę (o listach niebawem). Tutaj zwróci listę ['-','1.2','-','10','-'])
- **int**(waga), **float**(waga) (Zwraca liczbę (int/float) z łańcucha. Tutaj 37(int) lub 37.0 (float))

Inne ciekawe o łańcuchach

Łańcuch *napis* jest długości 8 i kolejne znaki w *napis* są indeksowane od 0 do 7. Możemy je wywołać/odczytać:

- napis[1] (Zwróci 'o')
- napis[-2] (Zwróci 'a')
- napis[2:4] (Zwróci 'j g')
- linia[2:11:3] (Zwróci łańcuch składający się z co 3 elementu zaczynając od znaku o indeksie 2 ale **mniejszych niż** 11 (czyli o indeksach 2,5,8 ale nie 11). Tutaj zwróci '1 1')

Co innego można ,,zrobić" z łańcuchem?

- **len**(linia) (Zwróci długość łańcucha linia)
- napis.**split(**'o'**)** (Podzieli zgodnie ze znakiem 'o'. Tutaj zwróci listę ['m','j graf'])
- 'af' **in** napis (Zwraca True lub False w zależności, czy 'af' jest w napis czy nie. Tutaj True.)
- 'af' **not in** napis (Tutaj zwraca False)
- napis+waga (Zwraca sklejony łańcuch. Tutaj zwraca 'moj graf37')
- napis+ ', ' + waga (Zwraca sklejony łańcuch. Tutaj zwraca 'moj graf, 37')
- napis.**find**('a') (Zwraca pozycję znaku w łańcuchu. Jeśli nie znajdzie znaku zwraca -1. Tu 6)
- **max**(napis), **min**(napis) (Zwraca odpowiednio, największy i najmniejszy element z łańcucha. Tutaj r i spacja)
- linia.**count**('-') (Zwraca liczbę wystąpień znaku. Tutaj będzie 3.)

LISTY

Listy są **uporządkowane**. Ich elementy są numerowane od **0**. Przykłady list:

- wierzch=['a','b','c','d','e'] (Uporządkowana lista wierzchołków grafu G. Wierzchołki są zapisywane jako łańcuchy.)
- kraw=[['a','b'],['a','d'],['b','c'],['b','d'],['c','d'],['c','e']] (Uporządkowana lista krawędzi grafu G. Tutaj krawędzie są zapisywane jako uporządkowane listy. Później zobaczymy, jak zapisać je jako nieuporządkowane zbiory)
- wagi=[1.2,10,1,8,7,12] (Uporządkowana lista wag krawędzi grafu G. Wagi są zapisywane jako float (pierwsza) lub int (pozostałe))

Operacje na listach, które na pewno się przydadzą:

- wierzch**[2]** (Zwróci element o indeksie 2 z listy. Tutaj 'c')
- wierzch**[-2]** (Zwróci przedostatni element listy (drugi od końca). Tutaj 'd')
- wierzch**[2]=**'f' (Zamienia element o danym indeksie na zadany za znakiem '=' element. Po tym wierzch wygląda tak: ['a', 'b', 'f', 'd', 'e'])
- **len**(kraw) (Zwraca liczbę elementów w liście. Tutaj 6)
- v=wierzch.**pop**(3) (Zwraca element o indeksie 3 i usuwa go z listy. Po tym v przyjmuje wartość 'd' oraz pierwotny wierzch wygląda tak: ['a','b', 'c', 'e'])
- **del** wierzch[3] (Usuwa element o indeksie 3 z listy. Po tym pierwotny wierzch wygląda tak: ['a','b', 'c', 'e'])
- nowa=wierzch.**copy()** lub nowa=**list(**wierzch**)** (Zwracają nową listę, o tych samych elementach co wierzch. UWAGA: Gdybyśmy zrobili nowa=wierzch, to jakakolwiek zmiana jednej listy pociąga automatycznie zmianę drugiej listy. Te komendy pozwalają tego uniknąć)
- nowa=wierzch+kraw (Tworzy nową listę, która jest połączeniem poprzednich. Tutaj nowa będzie wyglądać tak: ['a','b','c','d','e',['a','b'],['a','d'],['b','c'],['b','d'],['c','d'],['c','e']])
- wierzch.**append**('f') (Dodaje zadany element na koniec listy. Tutaj wierzch będzie wyglądać tak: ['a','b','c','d','e','f')
- wagi*2, [0]*6 (Zwraca zwielokrotnioną sklejoną listę. Tutaj: [1.2,10,1,8,7,12,1.2,10,1,8,7,12] oraz [0,0,0,0,0,0])

Operacje na listach, które może się przydadzą:

- wierzch**[1:3]** (Zwróci listę z elementami o indeksach 1 i 2 (bez 3!!!). Tutaj ['b','c'])
- wagi**[1:5:2]** (Zwróci listę z co 2 elementem o indeksach zaczynając od 1, ale mniejszych niż 5. Tutaj [10,8])
- 'a' **in** wierzch, 'a' **not in** wierzch (Zwraca True lub Falce, w zależności od tego, czy zadany element jest na liście czy nie. Tutaj zwraca, odpowiednio, True i False)

- wierzch.**insert**(2,'f') (Wstawia na zadane miejsce określony element. Po tym pierwotny wierzch wygląda tak: ['a', 'b', 'f', 'c', 'd', 'e'])
- wierzch.**remove**('b') (Usuwa **pierwsze wystąpienie** danego elementu. Po tym pierwotny wierzch wygląda tak: ['a', 'c', 'd', 'e']), BŁĄD gdy nie ma elementu na liście
- wierzch+=['f','g','h'] lub wierzch.**extend**(['f','g','h']) (Dodaje elementy z jednej listy do drugiej. Tutaj wierzch będzie wyglądać tak: ['a','b','c','d','e','f','g','h'])
- wierzch.**index**('c') (Zwraca indeks pierwszego wystąpienia elementu 'c'. Tutaj będzie 2)
- wagi.**sort**() (Sortuje elementy na liście. Tutaj wagi przyjmie wartość: [1.2, 1, 7, 8, 10, 12].)
- wagi.**count**(2) (Zwraca liczbę wystąpień określonego elementu. Tutaj zwraca 1.)
- **max**(wagi), **min**(wagi) (Zwraca, odpowiednio, największy i najmniejszy element z listy. Tutaj 12 i 1)

Jak zapisać macierz wag grafu G? Można ją zapisać jako listę list.

∞	1.2	∞	10	∞
1.2	∞	1	8	∞
∞	1	∞	7	12
10	8	7	∞	∞
∞	∞	12	∞	∞

```
inf=float('inf') #to dla wygody, żebyśmy nie musieli za dużo pisać
macierzwag=[[inf,1.2,inf,10,inf],[1.2,inf,1,8,inf],[inf,1,inf,7,12],[10,8,7,inf,inf]],
[inf,inf,12,inf,inf]]
```

Wtedy

macierzwag[i] jest listą elementów z wiersza i+1

macierzwag[i][j] jest elementem o indeksie j z listy wagi[i], czyli elementem z wiersza i+1 i kolumny j+1

UWAGA: Przypominamy, że elementy w listach są indeksowane od 0.

Analogicznie, jako listę list można zapisać: macierze przyległości, incydencji, oraz listy następników.

Jeszcze **LISTY SKŁADANE**. Przykłady:

```
pierwsze=[krawedz[0] for krawedz in kraw]
```
(pierwsze to lista pierwszych wierzchołków każdej z krawędzi na liście kraw, tzn. ['a','a','b','b','c','c'])

```
przekatna=[wagi[i][i] for i in range(5)]
```
(przekatna to lista wartości z przekątnej macierzy wag, tzn. [inf,inf,inf,inf,inf])
Wiersz macierzy przyległości linia=['0','1','1','0','1','0'] możemy szybko zamienić na wiersz int

```
liniaint=[int(i) for i in linia]
```
Wtedy liniaint jest postaci [0,1,1,0,1,0]

KROTKI

Krotki są niezmiennym odpowiednikiem list. Ich elementy są numerowane od **0**. Przykłady krotek:

- wierzchplus=('a','b','c','d','e')
- krawplus=(('a','b'),('a','d'),('b','c'),('b','d'),('c','d'),('c','e'))
- wagiplus=(1.2,10,1,8,7,12)

Co można zrobić z krotkami? Tylko niektóre z rzeczy, które można zrobić też na listach (te związane z ,,odczytywaniem'') i nic więcej (nic nie można ,,dodać'' ani ,,ująć'').

Co się przyda:

- wierzchplus**[2]** (Zwróci element 2 z krotki. Tutaj 'c')
- wierzchplus**[-2]** (Zwróci przedostatni element krotki 'd')

Co się może przyda:

- wierzchplus**[1:3]** (Zwróci kratkę z elementami o indeksach 1 i 2 (bez 3!!!). Tutaj ('b','c'))
- wagiplus**[1:5:2]** (Zwróci kratkę z co 2 elementem o indeksach zaczynając od 1, ale mniejszych niż 5. Tutaj (10,8))
- 'a' **in** wierzchplus, 'a' **not in** wierzchplus (Zwraca True lub Falce, w zależności od tego, czy zadany element jest w krotce czy nie. Tu zwraca, odpowiednio, True i False)
- **len**(krawplus) (Zwraca liczbę elementów w krotce. Tutaj 6.)
- nowa=wierzchplus+krawplus (Tworzy nową krotkę, która jest połączeniem poprzednich. Tutaj nowa będzie wyglądać tak: ('a', 'b', 'c', 'd', 'e', ('a', 'b'), ('a', 'd'), ('b', 'c'), ('b', 'd'), ('c', 'd'), ('c', 'e')).)
- wierzchplus.**index**('a') (Zwraca indeks pierwszego wystąpienia elementu 'a'. Tutaj będzie 0.)
- wagiplus.**count**(2) (Zwraca liczbę wystąpień określonego elementu. Tutaj zwraca 1.)
- **max**(wagiplus), **min**(wagiplus) (Zwraca, odpowiednio, największy i najmniejszy element z krotki. Tutaj 12 i 1.)

Z krotki można odzyskać listę, ale krotka pozostanie bez zmian:

- **list**(wagiplus) (Zwraca listę postaci: [1.2, 10, 1, 8, 7, 12], wagiplus pozostanie niezmieniona)

SŁOWNIKI

Słowniki są nieuporządkowane. Są **odwzorowaniami**, które kluczom (**key**) przyporządkowują pewne wartości (**value**). Za ich pomocą możemy na przykład zapisać listy następników grafu G.

nast={'a':['b','d'],'b':['a','d','c'],'c':['b','d','e'],'d':['a','b','c'],'e':['c']}

Wierzchołki są kluczami (**key**) a listy ich następników są wartościami (**value**) przypisanymi wierzchołkom.

Można też zapisać tak wagi krawędzi
wagikraw={('a','b'):1.2,('a','d'):10,('b','c'):1,('b','d'):8,('c','d'):7,('c','e'):12}
(UWAGA: Z natury zmienne listy/słowniki/zbiory nie mogą być kluczami. Dlatego wykorzystaliśmy tutaj krotki jako klucze.)

lub cały graf z wagami
graf={'a':{'b':1.2,'d':10},'b':{'a':1.2,'c':1,'d':8},'c':{'b':1,'d':7,'e':12},'d':{'a':10,'b':8,'c':7},'e':{'c':12}}

Operacje na słownikach, które na pewno się przydadzą:

- wagikraw[('a','b')] (Zwraca wartość o podanym kluczu. Tutaj zwróci 1.2.)
- wagikraw[('a','b')] = 7.7 (Zmienia wartość o zadanym kluczu. Teraz wagikraw będzie wyglądał tak: {('a','b'):7.7,('a','d'):10,('b','c'):1,('b','d'):8,('c','d'):7,('c','e'):12}.)
- wagikraw[('b','e')]=100 (Dodaje wartość 100 o kluczu ('b','e'). Teraz wagikraw będzie wyglądał tak: {('a','b'):1.2,('a','d'):10,('b','c'):1,('b','d'):8,('c','d'):7,('c','e'):12, ('b','e'):100}.)
- **del** wagikraw[('a','b')] (Usuwa element o zadanym kluczu. Teraz wagikraw wygląda tak: {('a','d'):10,('b','c'):1,('b','d'):8,('c','d'):7,('c','e'):12}.)
- wagikraw.**values**() (Zwraca wartości słownika. Można wykorzystać w pętli for.)
- wagikraw.**keys**() (Zwraca klucze słownika. Można wykorzystać w pętli for.)

Operacje na słownikach, które może się przydadzą:

- lista=**list**(wagikraw.**values**()) (Zwraca listę składającą się z wartości ze słownika. Tutaj lista jest postaci [1.2,10,1,8,7,12].)
- lista=**list**(wagikraw.**keys**()) (Zwraca listę składającą się z kluczy ze słownika. Tutaj lista jest postaci [('a','b'),('a','d'),('b','c'),('b','d'),('c','d'),('c','e')].)
- ('a','b') **in** wagikraw, ('a','b') **not in** wagikraw (Zwraca True lub False w zależności, czy dany element jest kluczem w słowniku. Tutaj zwraca True i False, odpowiednio.)

- len(wagikraw) (Zwraca liczbę kluczy w słowniku.)
- nowe=wagikraw.**pop**(('a','b')) (Zwraca wartość zadanego klucza i usuwa element o tym kluczu. Teraz nowe jest równe 1.2 a wagikraw wygląda tak: {('a','d'):10,('b','c'):1,('b','d'):8, ('c','d'):7,('c','e'):12}.)
- nowy=wagikraw.**copy**() lub nowy=**dict**(wagikraw) (Tworzą nowy słownik, o tych samych elementach co wagikraw. UWAGA: Gdybyśmy zrobili nowy=wagikraw, to jakakolwiek zmiana jednego słownika pociąga automatycznie zmianę drugiego słownika. Te komendy pozwalają tego uniknąć.)
- **min**(wagikraw), **min**(wagikraw.values()) (Zwrócą, odpowiednio, minimalny klucz i wartość, **max()** analogicznie. Jak wyznaczyć krawędź o minimalnej wadze opisane jest w rozdziale o lambda.)

UWAGA: Jak odczytać/zmienić/dopisać wagę krawędzi ze słownika graf?

- graf['a']['b'] (Zwróci wagę krawędzi ab)
- graf['a']['e']=20
  graf['e']['a']=20
  (Zmieni graf na {'a': {'b': 1.2, 'd': 10, 'e': 20}, 'b': {'a': 1.2, 'c': 1, 'd': 8}, 'c': {'b': 1, 'd': 7, 'e': 12}, 'd': {'a': 10, 'b': 8, 'c': 7}, 'e': {'c': 12, 'a': 20}}, tzn. doda krawędź ae o wadze 20. Ważne jest, żeby zmienić/dodać wagę na obu końcach krawędzi, jeśli graf nie jest skierowany.)
- graf['a']['d']=17
  graf['d']['a']=17
  (Zmieni graf na {'a': {'b': 1.2, 'd': 17}, 'b': {'a': 1.2, 'c': 1, 'd': 8}, 'c': {'b': 1, 'd': 7, 'e': 12}, 'd': {'a': 17, 'b': 8, 'c': 7}, 'e': {'c': 12}}, tzn. zmieni wagę krawędzi ad na 17.)
- del graf['a']['b']
  del graf['b']['a']
  (Usunie z grafu krawędź ab z jej wagą. Zmieni pierwotny graf na {'a': {'d': 10}, 'b': {'c': 1, 'd': 8}, 'c': {'b': 1, 'd': 7, 'e': 12}, 'd': {'a': 10, 'b': 8, 'c': 7}, 'e': {'c': 12}})

ZBIORY

Zbiory są nieuporządkowane. Przykłady:

zbwierzch={'a','b','c','d','e'}
zbkraw={('a','b'),('a','d'),('b','c'),('b','d'),('c','d'),('c','e')}
(nie można zrobić zbioru zbiorów, ale można zrobić listę zbiorów)
listakraw=[{'a','b'},{'a','d'},{'b','c'},{'b','d'},{'c','d'},{'c','e'}]

Zbiorów raczej nie będziemy używać. Operacje na zbiorach:

- **len**(zbwierzch) (Zwraca długość. Tutaj 5.)
- zbwierzch.**remove**('a'), zbwierzch.**discard**('a') (Usuwa element. Druga komenda nie zwraca błędu, gdy elementu nie ma w zbiorze.)
- zbwierzch.**union**(zbkraw) (Zwraca sumę zbiorów, zbwierzch i zbkraw pozostają bez zmian. Tutaj zwraca {('a', 'b'), ('b', 'd'), 'e', ('a', 'd'), 'd', 'c', ('c', 'e'), ('b', 'c'), ('c', 'd'), 'b'}.)
- zbwierzch.**update**(zbkraw) (Dokłada elementy ze zbkraw do zbioru zbwierzch. Po tej operacji znwierzch wygląda tak:{('a', 'b'), ('b', 'd'), 'e', 'a', ('a', 'd'), 'd', 'c', ('c', 'e'), ('b', 'c'), ('c', 'd'), 'b'}.)
- zbwierzch.**add**('f') (Dodaje element do zbioru. Jeśli element już był, to zbiór się nie zmieni. Po tej operacji zbwierzch wygląda tak: {'e', 'a', 'f', 'd', 'c', 'b'})
- nowy=zbwierzch**.copy**() (Robi kopię zbioru. Tzn. nowy jest taki jak zbwierzch, ale zmiany jednego nie wpływa na zmianę drugiego.)
- 'a' **in** zbwierzch, 'a' **not in** zbwierzch (Zwraca True lub Falce, w zależności od tego, czy zadany element jest w zbiorze czy nie. Tu zwraca, odpowiednio, True i False)

TWORZENIE PUSTYCH OBIEKTÓW:

- zmienna=None
  (None ma swój własny typ NoneType. Nie jest ani liczbą 0, ani False, ani pustym łańcuchem)
- pustylancuch=''
- pustalista=list() lub pustalista=[]
- pustyslownik=dict() lub pustyslownik={}
- pustyzbior=set()

LISTY LIST I SŁOWNIKI SŁOWNIKÓW - JAK UŻYWAĆ

Jeśli
SLOWNIK={1:[3,4,5] , 2:[5], 3:[1,5], 4:[1], 5:[1,2,3]}
to
SLOWNIK[1] to jest wartość przypisana kluczowi 1, czyli lista: [3,4,5]
SLOWNIK[3] to jest wartość przypisana kluczowi 3, czyli lista: [1,5]
Zatem
SLOWNIK[1][0], to jest element o indeksie 0 z listy SLOWNIK[1], czyli: 3
SLOWNIK[1][2], to jest element o indeksie 2 z listy SLOWNIK[1], czyli: 5
Do tych list można dodawać element, np.
SLOWNIK[1].append(7) da nam słownik {1:[3,4,5,7] , 2:[5], 3:[1,5], 4:[1], 5:[2,3]}
albo
SLOWNIK[5]+=[8,9,10] da nam słownik {1:[3,4,5] , 2:[5], 3:[1,5], 4:[1], 5:[1,2,3,8,9,10]}
Natomiast
SLOWNIK[4][0]=19 zamieni wartość zapisaną na liście SLOWNIK[4] na miejscu o indeksie 0 na 19, czyli:{1:[3,4,5] , 2:[5], 3:[1,5], 4:[19], 5:[1,2,3,8,9,10]}

Inny słownik: E={(1,2):7, (1,5):9, (3,4): 8}.
W dwóch przypadkach będziemy robić pętlę po kluczach takiego słownika: for e in E
Jeśli w iteracji e=(1,2) to e[0] jest równe 1 oraz e[1] jest 2, oraz E[e] będzie równe 7
(uwaga : e to jest krotka, czyli podobna do listy i elementy możemy odczytać po indeksach),
Jeśli e=(3,4) , to e[0] jest równe 3 oraz e[1] jest 4,oraz E[e] będzie równe 8

A teraz przyjrzyjmy się macierzom:
M=[[1,2,3],[4,5,6],[7,8,9]], czyli macierz:

1	2	3
4	5	6
7	8	9

M[0] to lista [1,2,3], M[1] to lista [4,5,6] oraz M[2] to lista [7,8,9]
M[0][1] to element o indeksie 1 z listy M[0], czyli: 2
M[1][2] to element o indeksie 2 z listy M[1], czyli: 6…
oraz
M[1][2]=20 zamieni element o indeksie 2 z listy M[1] na 20, czyli M=[[1,2,3],[4,5,20],[7,8,9]]

No i słownik słowników:

G={1:{3:100,4:101} , 2:{5:103}, 3:{1:100,5:200}, 4:{1:101}, 5:{2:103,3:200}}
Zupełnie analogicznie:
G[1] jest wartością słownika G o kluczu 1, czyli słownikiem {3:100,4:101}
G[5] jest wartością słownika G o kluczu 5, czyli słownikiem {2:103,3:200}
oraz
G[1][3] jest wartością słownika G[1] o kluczu 3, czyli: 100
G[5][3] jest wartością słownika G[5] o kluczu 3, czyli: 200
Jak poprzednio
G[1][7]=300 doda do słownika G[1] klucz 7 o wartości 300, czyli
{1:{3:100,4:101,7:300} , 2:{5:103}, 3:{1:100,5:200}, 4:{1:101}, 5:{2:103,3:200}}
natomiast
G[5][3]=500 zamieni wartość klucza 3 w słowniku G[5] na 500, czyli
{1:{3:100,4:101} , 2:{5:103}, 3:{1:100,5:200}, 4:{1:101}, 5:{2:103,3:500}}

PRINT i INPUT

- **print**('Lubie grafy') (Napisze, to co w nawiasie bez ". Domyślnie przechodzi do kolejnej linii.)
- **print**('Lubie grafy', **end=**' ') (Napisze, to co w " i zakończy znakiem określonym przez end=. Tutaj po prostu nie przejdzie do kolejnej linii)
- **print**('Ala','ma','kota.') (Wypisze łańcuchy domyślnie je oddzielając spacją: Ala ma kota)
- **print**('Ala','ma','kota.',**sep=**',') (Wypisze łańcuchy oddzielając znakiem określonym przez sep=. Tutaj: Ala,ma,kota.)
- **print**('Ala','ma','kota.',**sep='\n'**) (Wypisze każdy łańcuch w oddzielnej linii. '\n' - znak końca linii.)

Jak wypisać elementy z listy/słownika? Znak ***** pozwala ,,rozpakować'' listę/słownik

- **print**(wierzch) **print**(*****wierzch) **print**(*****wierzch,**sep=**',')
- **print**(wagikraw)
- **print**(*wagikraw.keys(),sep=';') lub **print**(*wagikraw, sep=';')
- **print(***wagikraw.values(),**sep='**')**
- **print(**zbwierzch**), print**(*zbwierzch,**sep**=' :) ')

  Wypiszą, odpowiednio:

  ['a', 'b', 'c', 'd', 'e'] a b c d e a,b,c,d,e
  {('a', 'b'): 1.2, ('a', 'd'): 10, ('b', 'c'): 1, ('b', 'd'): 8, ('c', 'd'): 7, ('c', 'e'): 12}
  ('a', 'b');('a', 'd');('b', 'c');('b', 'd');('c', 'd');('c', 'e')
  1.2**10**1**8**7**12
  {'a', 'c', 'b', 'e', 'd'} a :) e :) c :) b :) d

Jak wypisać łańcuch, liczby i inne obiekty razem? Przypomnijmy, że napis='moj graf', lwierzch=100 i lkraw=200.

- **print**(napis**,**'ma'**,**lwierzch**,**'wierzcholkow i'**,**lkraw**,**'krawedzi') - tutaj kolejne różnorodne ,,fragmenty" tekstu oddzielamy przecinkami;
- **print**('**{}** ma **{}** wierzcholkow i **{}** krawedzi'**.format**(napis,lwierzch,lkraw)) – tutaj wplatamy w tekst inne obiekty/zmienne za pomocą {} a następnie wypisujemy ich listę w kolejności.

W obu przypadkach dostaniemy jako wynik wypisanie:

moj graf ma 100 wierzcholkow i 200 krawedzi

ale druga metoda wydaje się bardziej przejrzysta w zapisie, gdy wplatamy dużo zmiennych/obiektów.

- **input**('Podaj liczbę wierzchołków:') (Wczytuje kolejny wiersz standardowych danych wejściowych jako łańcuch znaków, czekając, jeśli żaden wiersz nie jest teraz dostępny. Zdanie ,,zachęty" jest opcjonalne.)

ODCZYTYWANIE DANYCH Z PLIKU TXT

Przykład zawartości pliku matrix.txt (macierz wag, - oznacza ∞):

- 1.2 – 10 -
1.2 – 1 8 -
- 1 – 7 12
10 8 7 - -
- - 12 - -

f = **open**('matrix.txt') (Otwiera plik matrix.txt – **domyślnie jako do czytania**.)

f.**readline**() - odczyta kolejną (zacznie od pierwszej) linię z pliku (jako łańcuch)

Plik tekstowy jest zapamiętany w liniach. Każda linia jest łańcuchem. Można po nich iterować:

```
macierz=[]
for linia in f:
    macierz+=[linia]
```

Lista macierz jest wtedy postaci

['- 1.2 – 10 -\n', '1.2 – 1 8 -\n', '- 1 – 7 12\n', '10 8 7 - - \n', '- - 12 - -']

(znak \n jest znakiem końca linii)

```
macierz=[]
for linia in f:
    macierz+=[linia.split()]
```

Lista macierz jest wtedy postaci:

[['-', '1.2', '–', '10', '-'], ['1.2', '–', '1', '8', '-'], ['-', '1', '–', '7', '12'], ['10', '8', '7', '-', '-'], ['-', '-', '12', '-', '-']]

lub (jeśli ostatnia linia kończy się znakiem przejścia do kolejnej linii - enter)

[['-', '1.2', '–', '10', '-'], ['1.2', '–', '1', '8', '-'], ['-', '1', '–', '7', '12'], ['10', '8', '7', '-', '-'], ['-', '-', '12', '-', '-'], []]

```
macierz=[]
for linia in f:
    macierz+=[linia.split(' ')]
```

Lista macierz jest wtedy postaci:

[['-', '1.2', '–', '10', '-\n'], ['1.2', '–', '1', '8', '-\n'], ['-', '1', '–', '7', '12\n'], ['10', '8', '7', '-', '-', '\n'], ['-', '-', '12', '-', '-']]

f.**close**() (Zamyka plik.)

FUNKCJE

def nazwa_funkcji(parametry funkcji podane kolejno po przecinkach)**:**

lista komend, które działają na parametrach

return *zmienna/obiekt/wartość, którą chcemy zwrócić*

Funkcje też mogą działać bez parametrów lub nic nie zwracać. W głównej części programu zapisujemy:
nowy=nazwa_funkcji(parametry_funkcji), gdy funkcja coś zwraca i chcemy tą wartość/obiekt zapisać jako nowy lub nazwa_funkcji(parametry_funkcji), gdy chcemy tylko, by funkcja zadziałała.
UWAGA: Obiekty (zmienne, listy, łańcuchy) definiowane lokalnie wewnątrz funkcji nie są dostępne globalnie, poza funkcją. W dodatku jeśli jakiś obiekt zmienny (lista,słownik) zostanie zmieniony wewnątrz funkcji, to zostanie z tymi zmianami po zakończeniu działania funkcji.
W analogicznej sytuacji, obiekt niezmienny (np. obiekt typu liczba, łańcuch) nie zmieni się.

Przykłady:

```
def lalka(taka):
    taka+=[1]
lista=[2,2]
lalka(lista)
print(lista)
```

→ Napisze [2,2,1] , bo lista została zmieniona w trakcie działania funkcji lalka().

```
def lalka(taka):
    taka+=1
liczba=7
lalka(liczba)
print(liczba)
```

→ Napisze 7, bo liczba nie zmieniła się mimo działania funkcji lalka().

```
def lalka(taka):
    taka+=1
    return taka
liczba=7
liczba=lalka(liczba)
print(liczba)
```

→ Napisze 8.

LAMBDA

Lambda jest krótką, anonimową funkcją.

- **lambda** *parametry wypisane po przecinkach : jak działa funkcja na parametrach*
- *x=**lambda** parametry wypisane po przecinkach : jak działa funkcja na parametrach*

  (Druga komenda przypisuje tej funkcji nazwę x(), choć przypisanie nazwy nie jest zazwyczaj konieczne.)

Przykłady zastosowania lambda:

- Znalezienie krawędzi o minimalnej wadze w słowniku wagikraw:

  min(wagikraw, **key=lambda** x : wagikraw[x]**)**

  Znajdujemy minimalny z wagikraw, ale kluczem (key=) porządkującym elementy w wagikraw są wartości funkcji lambda, czyli porządkujemy elementy x zgodnie z wartościami wagikraw[x] (wagiami krawędzi x).
  Tutaj zwróci (‘b’,’c’).
- Posortowanie listy kraw zgodnie z wagami krawędzi z wykorzystaniem słownika graf:

  kraw.**sort(key=lambda** x: graf[x[0]][x[1]]**)**

  x są kolejnymi krawędziami z listy kraw (te krawędzie to listy dwuelementowe). x[0] to pierwszy wierzchołek krawędzi x a x[1] to drugi wierzchołek. Zatem zgodnie z definicją graf[x[0]][x[1]] jest wagą krawędzi x. Zatem sortujemy krawędzie, ale zgodnie z ich wagami zapisanymi w graf. Po posortowaniu kraw wygląda tak:

  [['b', 'c'], ['a', 'b'], ['c', 'd'], ['b', 'd'], ['a', 'd'], ['c', 'e']]

SŁOWNIK

Niektórzy uważają, że to ,,nieelegancko'' nadawać polskie nazwy zmiennym/obiektom. Dla tych osób poniżej krótki słownik polsko-angielski terminów teoriografowych:

cykl – cycle
cykl Hamiltona – Hamilton cycle
drzewo – tree
drzewo rozpinające – spanning tree
drzewo o rozpinające o minimalnej wadze – minimum spanning tree (MST)
etykiety – labels
graf – graph
graf dwudzielny – bipartite graph
graf skierowany/digraf – digraph
incydentny – incident
kolor – color (lub colour dla zagorzałych ,,Brytoli'')
krawędź – edge
las – forest
lista następników – adjacency list
macierz incydencji – incidence matrix
macierz przyległości – adjecency matrix
macierz wag – weight matrix
mutligraf – multigraph
podgraf – subgraph
przyległy – adjacent
sąsiad – neighbor (lub neighbour dla zagorzałych ,,Brytoli'')
składowa spójności – connected component
skojarzenie - matching
spójny – connected
ścieżka – path
szlak/obchód Eulera - Eulerian trail/circuit
(najkrótsza) ścieżka - (shortest) path
waga - weight
wierzchołek (wierzchołki) – vertex (vertices), node (nodes)