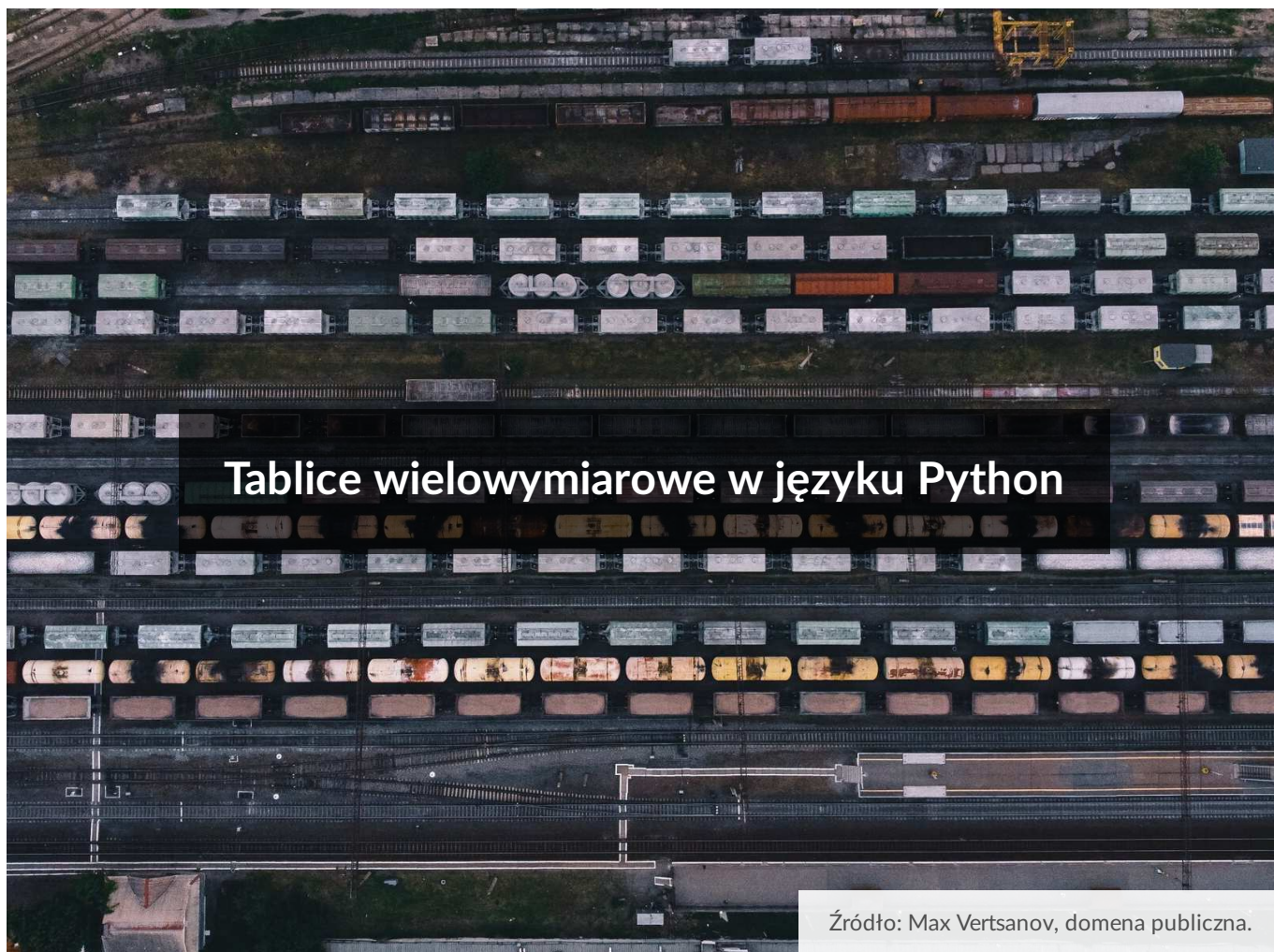




## Tablice wielowymiarowe w języku Python

- [Wprowadzenie](#)
- [Przeczytaj](#)
- [Aplet](#)
- [Sprawdź się](#)
- [Dla nauczyciela](#)



W tym e-materiale powtarzamy wiadomości ze szkoły podstawowej.

W e-materiale [Tablice wielowymiarowe](#) poznaliśmy definicję tego typu tablic. Czy potrafisz wskazać ich przykłady w życiu codziennym?

Jeśli nie, przypomnij sobie popularną grę w statki. Plansza jest niczym innym jak właśnie przykładem tablicy wielowymiarowej, a dokładniej dwuwymiarowej. Składa się ze 100 pól – każde z nich jest oznaczone przez jedną literę oraz jedną liczbę i przechowuje pewną informację, w tym przypadku o obecności lub nieobecności statku.

W tym e-materiale zapoznamy się z implementacją tablic wielowymiarowych w języku Python.

Implementacje tablic wielowymiarowych w innych językach programowania zostały omówione w e-materiałach:

- [Tablice wielowymiarowe w języku C++](#),
- [Tablice wielowymiarowe w języku Java](#).

Więcej zadań? Przejdź do e-materiału [Tablice wielowymiarowe – zadania maturalne](#).

Informacje na temat tablic jednowymiarowych znajdziesz w e-materiałach:

- [Tablice jednowymiarowe](#),
- [Tablice jednowymiarowe w języku Python](#).

### **Twoje cele**

- Użyjesz pętli zagnieżdżonych w języku Python.
- Zbudujesz tablice dwuwymiarowe (listy list).
- Wykonasz proste operacje na macierzach.
- Przeanalizujesz sposoby operowania danymi na listach wielowymiarowych.

# Przeczytaj

---

## Tablice dwuwymiarowe: definiowanie macierzy i operacje na ich elementach

Założmy, że gramy w statki. Pusta plansza do gry w statki liczy 100 pól.

	1	2	3	4	5	6	7	8	9	10
A										
B										
C										
D										
E										
F										
G										
H										
I										
J										

Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Wypełniliśmy już swoją planszę, zaznaczając na niej nasze okręty.

	1	2	3	4	5	6	7	8	9	10
A	■	■						■		
B								■		
C		■	■	■	■			■		
D								■		
E				■				■		
F				■						■
G				■						■
H										■
I						■	■			■
J		■	■	■						■

Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

Część pól dalej jest pusta, część została zaczerzerniona (znajdują się na nich okręty). Łatwo sprawdzić, czy dane pole zawiera statek czy nie – nazwa pola składa się z litery i liczby, np. A6.

Plansza składa się z kolumn oraz wierszy. Możemy stwierdzić, że składa się z kilku tablic (uporządkowanych zbiorów elementów tego samego typu) – jest tablicą tablic, czyli tablicą, które elementami są inne tablice.

W języku Python struktury danych, którymi są tablice, będziemy implementować za pomocą list. W naszym przypadku będzie to [lista wielowymiarowa](#).

W tym e-materiale mówiąc o tablicach, mamy na myśli strukturę danych, która w języku Python implementowana jest za pomocą listy.

### Ważne!

Pamiętaj, że w przypadku tablic wszystkie elementy muszą być tego samego typu, natomiast w przypadku list takie ograniczenie już nie obowiązuje. Więcej informacji na temat tablic i list znajdziesz w e-materiałach:

- [Podstawowe struktury danych: tablica](#),
- [Tablice jednowymiarowe w języku Python](#),
- [Tablice wielowymiarowe](#).

Aby utworzyć tablicę wielowymiarową w języku Python, możemy posłużyć się zapisem:

```
1 tablica_w = [ [element11, element12, element13],
2               [element21, element22, element23],
3               [element31, element32, element33],
4               ]
```

Ma ona trzy elementy – każdy z nich jest kolejną tablicą, która również zawiera trzy elementy. Warto jednak podkreślić, że każda z tablic mogłaby mieć dowolną długość.

### Ważne!

Zauważmy, że na końcu wiersza numer 3 znajduje się przecinek, pomimo braku kolejnego elementu. Jest to poprawny zapis w języku Python. Dzięki takiej notacji, rozbudowując program, możemy łatwo dodawać kolejne wiersze w tablicy.

Naszą planszę do gry w statki możemy przechować w programie w następujący sposób:

```
1 tablica_w = [
2     [1, 1, 0, 0, 0, 0, 0, 1, 0, 0 ],
3     [0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ],
```

```

4      [0, 1, 1, 1, 1, 0, 0, 1, 0, 0 ],
5      [0, 0, 0, 0, 0, 0, 0, 1, 0, 0 ],
6      [0, 0, 0, 1, 0, 0, 0, 1, 0, 0 ],
7      [0, 0, 0, 1, 0, 0, 0, 1, 0, 1 ],
8      [0, 0, 0, 1, 0, 0, 0, 0, 0, 1 ],
9      [0, 0, 0, 0, 0, 0, 0, 0, 0, 1 ],
10     [0, 0, 0, 0, 0, 1, 1, 0, 0, 1 ],
11     [0, 1, 1, 1, 0, 0, 0, 0, 0, 0 ]
12 ]

```

Tablicę możemy wypełnić danymi w różny sposób, m.in. przez:

1. przypisanie wartości na etapie jej inicjowania,
2. utworzenie nowej tablicy na podstawie innej tablicy za pomocą [wyrażeń indeksujących](#),
3. wykorzystanie generatora,
4. wstawianie elementów do tablicy, np. podanych przez użytkownika wyników obliczeń.

#### Ważne!

Odwołując się do konkretnego elementu w przypadku tablic dwuwymiarowych, podajemy dwa indeksy w sąsiadujących nawiasach kwadratowych. Przykładowo, zapis `tablica_glowna[i][j]` oznacza element o indeksie `j` w tablicy, która jest `i`-tym elementem tablicy `tablica_glowna`.

Oto przykład zdefiniowania tablicy wielowymiarowej i sposób przywoływania jej elementu:

```

1  tablica_glowna = [['a', 'b', 'c'],
2                   ['d', 'e', 'f'],
3                   ['g', 'h', 'i']]
4
5  print(tablica_glowna[0])
6  # Wynik: ['a', 'b', 'c']
7
8  print(tablica_glowna[0][0])
9  # Wynik: 'a'
10
11 print(tablica_glowna[0][2])
12 # Wynik: 'c'
13
14 print(tablica_glowna[2])
15 # Wynik: ['g', 'h', 'i']
16
17 print(tablica_glowna[2][1])

```

```
18 # Wynik: 'h'
19
20 print(tablica_glowna[2][2])
21 # Wynik: 'i'
```

### Ważne!

Jeżeli odwołamy się do elementu tablicy dwuwymiarowej, podając tylko jeden indeks, otrzymamy tablicę jednowymiarową (która jest elementem tablicy dwuwymiarowej). Podanie indeksu wykraczającego poza zakres dowolnej tablicy kończy się wyświetleniem komunikatu o błędzie: `IndexError: list index out of range`.

Pamiętajmy, że w języku Python występują ujemne indeksy, które wskazują na elementy tablicy, patrząc od jej prawej strony (od tyłu).

Wartości poszczególnych elementów możemy modyfikować w prosty sposób – przez przypisanie. Oto przykład takiej operacji:

```
1 tablica_a = [1, 2, 3]
2 tablica_b = [33, 23, 13, 45, 56]
3
4 print(tablica_a[2])
5 # Wynik: 3
6
7 # Tu przypisujemy wartość do elementu
8 tablica_a[2] = 999
9
10 print(tablica_a[2])
11 # Wynik: 999
12
13 print(tablica_b[3])
14 # Wynik: 45
15
16 # Tu przypisujemy wartość do elementu
17 tablica_b[3] = 2020
18 print(tablica_b[3])
19 # Wynik: 2020
20
21 print(tablica_b)
22 # Wynik: [33, 23, 13, 2020, 56]
```

Przeanalizujmy inny przypadek: z dwóch tablic jednowymiarowych o takiej samej liczbie elementów zbudujemy trzecią – dwuwymiarową. Zrobimy to na dwa sposoby:

- przez proste przypisanie tablic jako elementów,
- przez przypisanie kopii tablic jako elementów dzięki mechanizmowi wyrażeń indeksujących (`slice`).

```
1 tablica_a = [1, 2, 3]
2 tablica_b = [33, 23, 13]
3 tablica_duza = [lista_a, lista_b]
4 tablica_wielka = [lista_a[:], lista_b[:]]
5 tablica_a[2] = 999
6 tablica_b[1] = 2002
7
8 print(tablica_duza)
9 # Wynik: [[1, 2, 999], [33, 2002, 13]]
10
11 print(tablica_wielka)
12 # Wynik: [[1, 2, 3], [33, 23, 13]]
```

### Dla zainteresowanych

Funkcja `del()` kasuje nieodwracalnie wskazane indeksem elementy bądź całą strukturę. Więcej na jej temat można przeczytać w dokumentacji języka Python 3.3.

Do dyspozycji mamy też metody:

- `pop(index)` – zwraca wartość elementu o numerze `index` i kasuje ten obiekt,
- `remove(wartość)` – kasuje pierwszy element o podanej wartości.

### Przykład 1

Napiszmy program, który wypisze na ekranie tabliczkę mnożenia dla czynników z przedziału prawostronnie otwartego od początkowy do koncowy. Przyjmijmy założenie, że liczba elementów w wierszu i w kolumnie będzie identyczna:

#### Specyfikacja problemu:

*Dane:*

- `początkowy` – początek przedziału
- `koncowy` – koniec przedziału

Wynik:

Program wypisuje na ekran tabliczkę mnożenia dla czynników z przedziału prawostronnie otwartego od początkowy do koncowy.

```
1 początkowy = 1
2 koncowy = 21
3 tablica = [ ]
4
5 # tworzymy pierwszy wiersz, bazując na generatorze listowym
6 wiersz = [ '*' ] + [ x for x in range(początkowy, koncowy)]
7 dl_wiersz = len(wiersz)
8 liczba_elementow = koncowy - początkowy
9 tablica.append(wiersz)
10
11 # tworzymy odpowiednią liczbę wierszy (tablic jednowymiarowych)
12 for element in wiersz:
13     if element != '*':
14         nowy_wiersz = [ element ] + ( [ None ] * liczba_elementow)
15         tablica.append(nowy_wiersz)
16
17 # obliczamy wynik operacji (mnożenie) i wpisujemy do odpowiedni
18 # elementów tablic zagnieżdżonych
19 for i in range(1,dl_wiersz):
20     mnoznik1 = tablica[0][i]
21     for j in range(1,dl_wiersz):
22         mnoznik2 = tablica[j][0]
23         wynik = mnoznik1 * mnoznik2
24         tablica[j][i] = wynik
25
26 # wypisujemy wyniki na ekranie
27 print(' ')
28 for wiersz in tablica:
29     for element in wiersz:
30         print("{:>5}|".format(element),end="")
31     print(' ')
32
```

Rezultat działania funkcji wygląda następująco:



1	*	1	2	3	4	5	6	7	8	9	
2	1	1	2	3	4	5	6	7	8	9	
3	2	2	4	6	8	10	12	14	16	18	
4	3	3	6	9	12	15	18	21	24	27	
5	4	4	8	12	16	20	24	28	32	36	
6	5	5	10	15	20	25	30	35	40	45	
7	6	6	12	18	24	30	36	42	48	54	
8	7	7	14	21	28	35	42	49	56	63	
9	8	8	16	24	32	40	48	56	64	72	
10	9	9	18	27	36	45	54	63	72	81	
11	10	10	20	30	40	50	60	70	80	90	1
12	11	11	22	33	44	55	66	77	88	99	1
13	12	12	24	36	48	60	72	84	96	108	1
14	13	13	26	39	52	65	78	91	104	117	1
15	14	14	28	42	56	70	84	98	112	126	1
16	15	15	30	45	60	75	90	105	120	135	1
17	16	16	32	48	64	80	96	112	128	144	1
18	17	17	34	51	68	85	102	119	136	153	1
19	18	18	36	54	72	90	108	126	144	162	1
20	19	19	38	57	76	95	114	133	152	171	1
21	20	20	40	60	80	100	120	140	160	180	2

## Ciekawostka

Do operowania na elementach listy lub tablicy możemy wykorzystać też funkcję `enumerate(iter)`, która zwraca obiekt typu `tuple` (indeks oraz wartość danego elementu), na przykład:

```

1 elementy = [ 'Python', 3.6, 'Linux', 'Windows', 'MacOS' ]
2 for ind , wart in enumerate(elementy):
3     print('Indeks', ind, 'Wartość', wart)
4
5 Indeks 0 Wartość Python
6 Indeks 1 Wartość 3.6
7 Indeks 2 Wartość Linux
8 Indeks 3 Wartość Windows
9 Indeks 4 Wartość MacOS
10
11 elementy_dwuwymiarowe = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
12 for ind , wart in enumerate(elementy_dwuwymiarowe):
13     print('Indeks', ind, 'Wartość', wart)

```

14	Indeks 0	Wartość [1, 2, 3]
15	Indeks 1	Wartość [4, 5, 6]
16	Indeks 2	Wartość [7, 8, 9]

## Operacje na macierzach

Tablice dwuwymiarowe mogą posłużyć do przeprowadzania operacji na [macierzach](#).

### Przykład 2



Napiszmy funkcję obliczającą sumę dwóch macierzy, pamiętając, że możemy dodawać tylko macierze o tych samych wymiarach.

Działanie programu przetestujemy dla macierzy:

```
1 A = [[1, 2], [3, 4]]
2 B = [[5, 6], [7, 8]]
```

### Specyfikacja problemu:

*Dane:*

- A - składnik dodawania; macierz kwadratowa rozmiaru  $n \times n$
- B - składnik dodawania; macierz kwadratowa rozmiaru  $n \times n$

*Wyniki:*

- C - wynik dodawania macierzy  $A + B$ ; macierz kwadratowa rozmiaru  $n \times n$

Oto dwie macierze (tablice/listy dwuwymiarowe):

### Macierz A

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Którą w kodzie możemy zapisać jako:

```
1 A = [[a, b], [c, d]]
```

### Macierz B

$$B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

Którą w kodzie możemy zapisać jako:

```
1 B = [[e, f], [g, h]]
```

### Macierz C

$$C = \begin{bmatrix} a + e & b + f \\ c + g & d + h \end{bmatrix}$$

Operacja dodawania dwóch macierzy  $C = A + B$  polega na dodawaniu elementów o takich samych indeksach, czyli  $C[i][j] = A[i][j] + B[i][j]$ .

Którą w kodzie możemy zapisać jako:

```
1 C = [[a + e, b + f], [c + g, d + h]]
```

Korzystając z takiej definicji, możemy zapisać funkcję w języku Python:

```
1 def suma_macierzy(A, B):
2     rozmiar_macierzy = len(A)
3     C = [[0 for i in range(rozmiar_macierzy)] for i in range(ro
4
5     for x in range(rozmiar_macierzy):
6         for y in range(rozmiar_macierzy):
7             C[x][y] = A[x][y] + B[x][y]
8
9     return C
```

Wynikiem jej działania jest macierz C będąca wynikiem operacji dodawania macierzy  $C = A + B$

```
1 A = [[1, 2], [3, 4]]
2 B = [[5, 6], [7, 8]]
3
4 print(suma_macierzy(A, B))
5 # Wynik: [[6, 8], [10, 12]]
```

Co w sytuacji, kiedy do czynienia mamy z macierzą prostokątną o wymiarach  $n \times m$ ?  
Zapoznaj się z implementacją.

```
1 def suma_macierzy(A, B):
2     n = len(A)
3     m = len(A[0])
4     C = [[0 for i in range(m)] for i in range(n)]
5
6     for x in range(n):
7         for y in range(m):
8             C[x][y] = A[x][y] + B[x][y]
9
10    return C
11
12 A = [[1, 2], [3, 4], [3, 4]]
13 B = [[5, 6], [7, 8], [7, 8]]
14
15 print(suma_macierzy(A, B))
```

## Przykłady operacji na tablicach n-wymiarowych

W języku Python nie ma ograniczeń w stosunku do liczby zagnieżdżonych tablic. Możemy z łatwością stworzyć tablicę trójwymiarową.

### Przykład 3



Oto przykład tablicy z trzema wymiarami, czyli tablicy zagnieżdżonej w tablicy, która jest zagnieżdżona w kolejnej tablicy:

```
1 lw = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
2
3 # przykłady wyświetlania poszczególny elementów wskazywanych po
4 print(lw[1])
5 [[5, 6], [7, 8]]
6
7 print(lw[0][1])
8 [3, 4]
9
10 print(lw[1][1][0])
11 7
```

Możemy również wykorzystać pętle, aby pokazać kolejne zagnieżdżenia:

```
1 lw = [[[1, 2], [3, 4]], [[5, 6], [7, 8]]]
2
3 for element in lw:
4     print(element, type(element))
5     print("Wewnątrz 1:")
6     for elem in element:
7         print(elem, type(elem))
8         print("=> Wewnątrz 2:")
9         for elem2 in elem:
10            print(elem2, type(elem2))
11
12 # efekt wykonania
13
14 [[1, 2], [3, 4]] <class 'list'>
15 Wewnątrz 1:
16 [1, 2] <class 'list'>
17 => Wewnątrz 2:
18 1 <class 'int'>
19 2 <class 'int'>
20 [3, 4] <class 'list'>
21 => Wewnątrz 2:
22 3 <class 'int'>
23 4 <class 'int'>
24 [[5, 6], [7, 8]] <class 'list'>
25 Wewnątrz 1:
26 [5, 6] <class 'list'>
27 => Wewnątrz 2:
28 5 <class 'int'>
29 6 <class 'int'>
30 [7, 8] <class 'list'>
31 => Wewnątrz 2:
32 7 <class 'int'>
33 8 <class 'int'>
```

### Dla zainteresowanych

Istnieje moduł o nazwie NumPy – umożliwia on wykonywanie operacji na tablicach liczb.

# Słownik

## lista wielowymiarowa

lista złożona z elementów będących listami; w języku Python nie ma ograniczeń co do jej rozmiarów (poza ilością dostępnej pamięci)

## macierz

zbiór liczb lub wyrażeń zapisanych w postaci prostokątnej tablicy

## obiekt iterowany

(ang. *iterable*) obiekt, który ma budowę sekwencyjną; może być przekazywany do pętli `for` w celu zwracania pojedynczych elementów większej całości; obiektami iterowanymi są `list`, `string`, `tuple`, np. `[1, 3, 5]` lub `'Python'` lub `(3, 6, 'A')`

## wyrażenie indeksujące

(ang. *slice*) wyrażanie zwracające wycinek obiektu iterowanego (lista, krotka, napis, itp.), zapisywane za pomocą nawiasów kwadratowych

`obiekt_iterowany[start:stop:krok]`, które jako argumenty przyjmuje początek, koniec (wyłącznie) oraz krok wycinka; przykładowo wyrażenie `"Przykład"[0:4:2]` da wynik `Pz` (zwraca znaki pod indeksami 0 oraz 2)

# Aplet

---

## Polecenie 1

Uruchom aplet symulujący działanie notacji wycinkowej (ang. *slice notation*) dla tablicy dwuwymiarowej.

Przyjrzyjmy się przykładowi.

```
1 A[2::3, 9::-2]
```

Ogólny zapis notacji wycinków:

```
1 a[start:stop:krok]
```

W przykładzie widzimy jednak zapis z przecinkiem. Odnosi się on do tego, że do czynienia mamy z tablicą dwuwymiarową. Pierwsza część zapisu odnosi się do wierszy, druga do kolumn. Rozłóżmy ją tak, żeby była zrozumiała.

Elementy wycinka indeksujemy od wiersza o indeksie 2 aż do końca (zapis `::`).

Wybieramy co trzeci wiersz (krok wynosi 3).

Podobnie w przypadku kolumn. Zaczynamy od kolumny o indeksie 9, ale tym razem liczymy od końca do początku tablicy z krokiem 2.

Przetestuj różne możliwości – czy zawsze wiesz, jaki wynik otrzymasz?

Notacja wycinkowa:

A[2::3, 9::-2]

9	7	9	0	8	4	8	6	6	0
4	9	8	2	9	9	5	8	7	0
7	3	1	1	8	7	2	4	3	4
0	1	6	4	6	0	5	9	9	7
5	4	3	4	3	1	5	8	7	3
9	1	5	4	0	8	4	2	9	9
0	7	3	8	3	2	1	9	0	7
4	6	0	8	7	2	9	1	7	3
3	6	7	0	0	4	6	4	1	4
2	5	8	4	5	8	0	0	3	8

4	4	7	1	3
9	2	8	4	1
4	4	4	0	6




Zasób interaktywny dostępny pod adresem <https://zpe.gov.pl/a/Dq3bcGGJS>

Źródło: Contentplus.pl sp. z o.o., licencja: CC BY-SA 3.0.

## Polecenie 2

# Sprawdź się

---

Pokaż ćwiczenia:   

Ćwiczenie 1



Ćwiczenie 2



### Ćwiczenie 3



Dana jest tablica o wymiarach  $n \times n$ . Napisz program, który wypisze zawartość jedynie tych wierszy tablicy, w których wszystkie elementy są większe od liczby 2.

Działanie swojego programu przetestuj dla tablicy wypełnionej wartościami w następujący sposób: każda komórka tablicy przyjmuje wartość sumy obu jej indeksów, czyli  $tablica[i][j] = i + j$ . Rozwiązanie sprawdź dla  $n = 5$ .

#### Specyfikacja problemu:

*Dane:*

- `tablica` – tablica o wymiarach  $n \times n$
- `n` – liczba naturalna dodatnia; liczba wierszy i kolumn

*Wynik:*

Program wypisuje zawartość wierszy tablicy, w których wszystkie elementy są większe od 2.

*Przykładowe wyjście:*

```
1 3 4 5 6 7
2 4 5 6 7 8
```

## Ćwiczenie 4



Napisz program wypisujący kwadrat sumy wartości znajdujących się na obu przekątnych tablicy o wymiarach  $n \times n$  (wartości wspólne mogą się powtarzać).

Działanie swojego programu przetestuj dla tablicy wypełnionej wartościami w następujący sposób: każda komórka tablicy przyjmuje wartość sumy obu jej indeksów, czyli  $tablica[i][j] = i + j$ . Rozwiązanie sprawdź dla  $n = 5$ .

### Specyfikacja problemu:

*Dane:*

- `tablica` – tablica o wymiarach  $n \times n$
- `n` – liczba naturalna dodatnia; liczba wierszy i kolumn

*Wynik:*

Program wypisuje kwadrat sumy wartości znajdujących się na obu przekątnych danej tablicy.

*Przykładowe wyjście dla  $n = 5$ :*

```
1 1600
```

## Ćwiczenie 5



Dane są dwie tablice: `macierz_a`, `macierz_b` o wymiarach  $n \times m$ . Elementami obu tych tablic są liczby całkowite. Napisz program, który utworzy trzecią macierz - `macierz_wynik` o takich samych wymiarach i wypełni ją w ten sposób, by dla wszystkich  $i \in \langle 0, N \rangle, j \in \langle 0, M \rangle$  zachodziła równość:

$$macierz\_wynik[i][j] = \min(macierz\_a[i][j], macierz\_b[i][j])$$

Działanie swojego programu przetestuj dla następujących macierzy:

```
1 macierz_a = [[346, 654, 865, 278], [243, 765, 869, 114], [543,
2 macierz_b = [[123, 765, 867, 431], [356, 543, 235, 853], [649,
```

### Specyfikacja problemu:

*Dane:*

- `macierz_a` – tablica o wymiarach  $n \times m$
- `macierz_b` – tablica o wymiarach  $n \times m$
- `m` – liczba naturalna dodatnia, liczba kolumn
- `n` – liczba naturalna dodatnia, liczba wierszy

*Wynik:*

Program wypisze nową macierz wypełnioną zgodnie z poleceniem.

*Przykładowe wyjście dla danych testowych:*

```
1 123 654 865 278
2 243 543 235 114
3 543 535 654 424
```

# Dla nauczyciela

---

**Autor:** Adam Jurkiewicz

**Przedmiot:** Informatyka

**Temat:** Tablice wielowymiarowe w języku Python

**Grupa docelowa:**

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

**Podstawa programowa:**

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

2) do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

3) objaśnia dobrany algorytm, uzasadnia poprawność rozwiązania na wybranych przykładach danych i ocenia jego efektywność;

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres podstawowy. Uczeń:

1) projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje

poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);

2) do realizacji rozwiązań problemów prawidłowo dobiera środowiska informatyczne, aplikacje oraz zasoby, wykorzystuje również elementy robotyki;

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) projektuje i tworzy rozbudowane programy w procesie rozwiązywania problemów, wykorzystuje w programach dobrane do algorytmów struktury danych, w tym struktury dynamiczne i korzysta z dostępnych bibliotek dla tych struktur;

2) stosuje zasady programowania strukturalnego i obiektowego w rozwiązywaniu problemów;

3) sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

#### **Kształtowane kompetencje kluczowe:**

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

#### **Cele operacyjne (językiem ucznia):**

- Użyjesz pętli zagnieżdżonych w języku Python.
- Zbudujesz tablice dwuwymiarowe (listy list).
- Wykonasz proste operacje na macierzach.
- Przeanalizujesz sposoby operowania danymi na listach wielowymiarowych.

#### **Strategie nauczania:**

- konstruktywizm;
- konektywizm.

#### **Metody i techniki nauczania:**

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimediu i ćwiczeń interaktywnych;
- ćwiczenia praktyczne.

#### **Formy pracy:**

- praca indywidualna;

- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

### Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

### Przebieg lekcji

#### Przed lekcją:

1. Uczniowie powtarzają informacje na temat list, tablic jednowymiarowych oraz tablic wielowymiarowych.
2. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Tablice wielowymiarowe w języku Python”. Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj”.

#### Faza wstępna:

1. Chętna lub wybrana osoba referuje najważniejsze informacje dotyczące list, tablic jednowymiarowych oraz tablic wielowymiarowych.
2. Nauczyciel wyświetla temat i cele zajęć zawarte w sekcji „Wprowadzenie”. Następnie wspólnie z uczniami określa kryteria sukcesu.
3. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi wybranego ucznia lub uczniów o przedstawienie sytuacji problemowej związanej z tematem lekcji.

#### Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel sprawdza przygotowanie uczniów do lekcji. Jeśli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj”.
2. **Praca z multimediami.** Uczniowie testują działanie apletu z sekcji „Aplet”. W kolejnym kroku nauczyciel prosi uczniów o podanie innego przykładu zastosowania tablic (na podstawie przykładu gry w statki w sekcji „Przeczytaj”).
3. **Ćwiczenie umiejętności.** Uczniowie wykonują ćwiczenia nr 1–4 z sekcji „Sprawdź się”. Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

#### Faza podsumowująca:

1. Na koniec zajęć nauczyciel raz jeszcze wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie”. W odniesieniu do ich realizacji dokonuje szczegółowej oceny

rozwiązania zastosowanego przez wybranego ucznia.

- Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń z programowania w języku Python.

#### **Praca domowa:**

- Uczniowie wykonują ćwiczenie nr 5 z sekcji „Sprawdź się”.
- Uczniowie wykonują polecenie 2 z sekcji „Aplet”.

#### **Materiały pomocnicze:**

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

#### **Wskazówki metodyczne:**

- Aplet może zostać wykorzystany do zaprezentowania działania notacji wycinkowej.