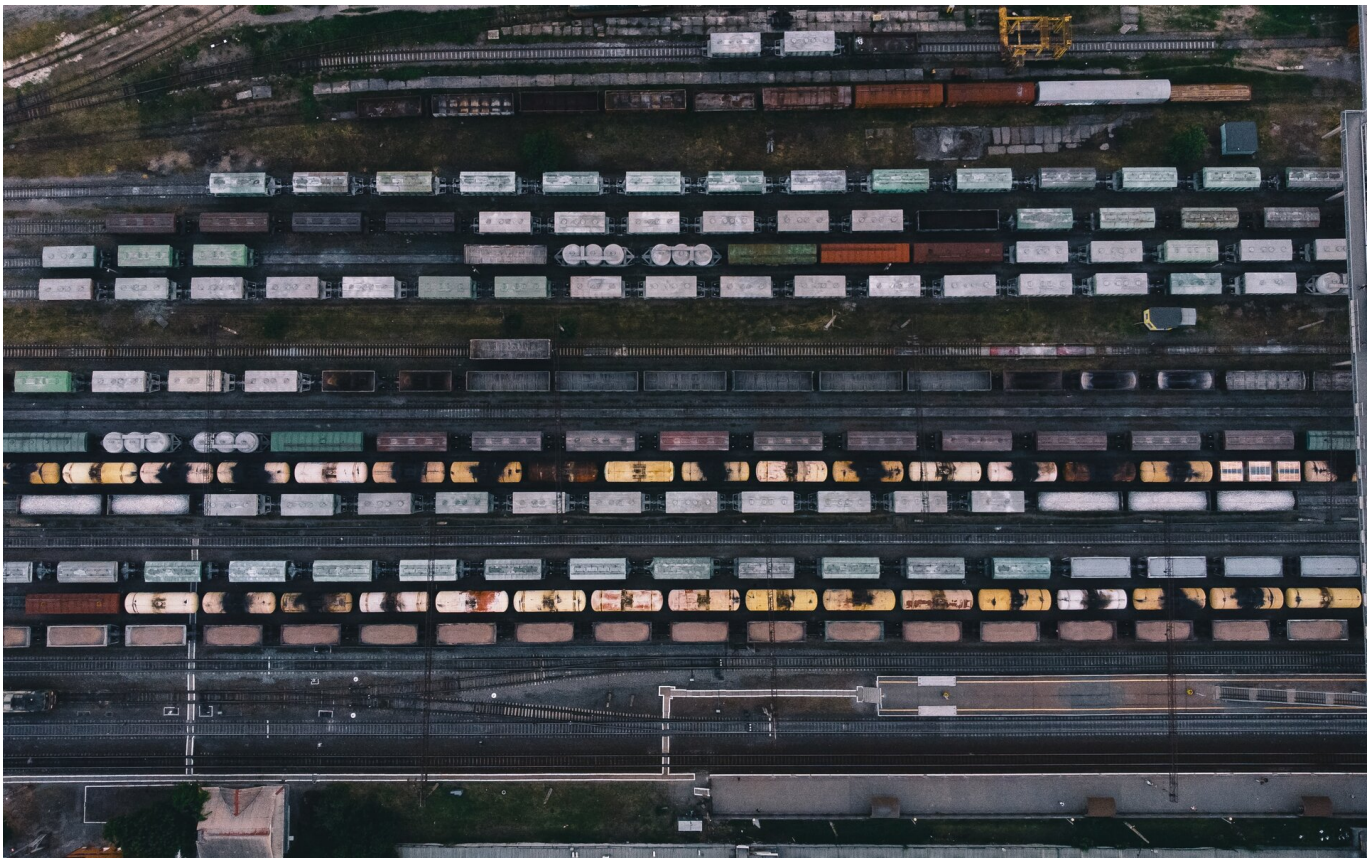
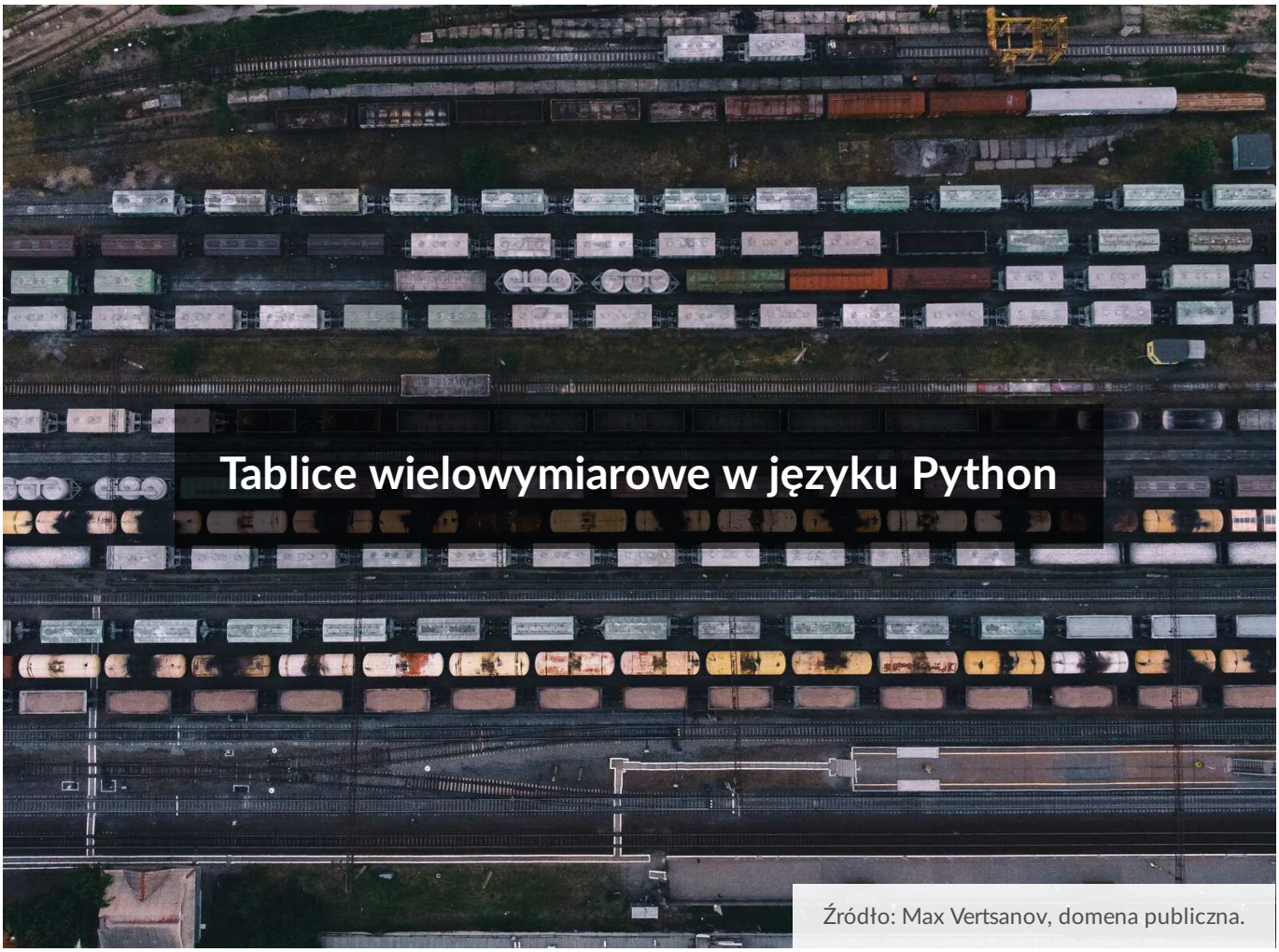


Tablice wielowymiarowe w języku Python



Źródło: Max Vertsanov, domena publiczna.

> 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.

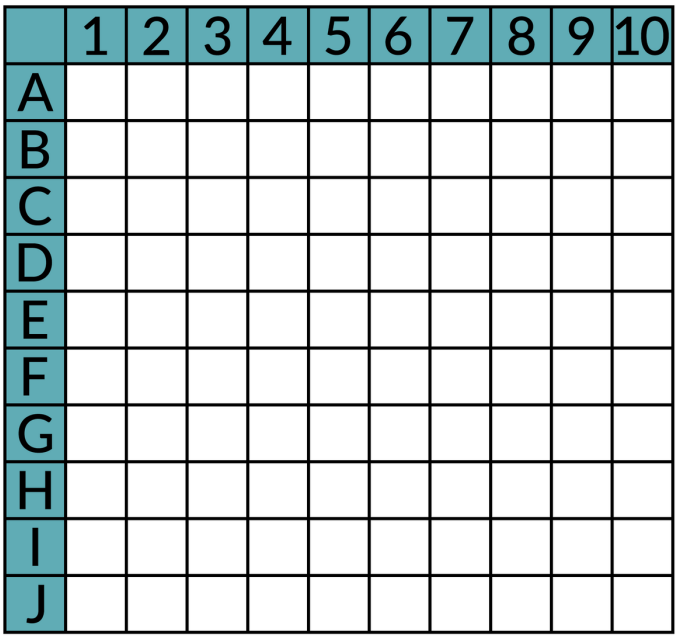
Twoje cele

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

Przeczytaj

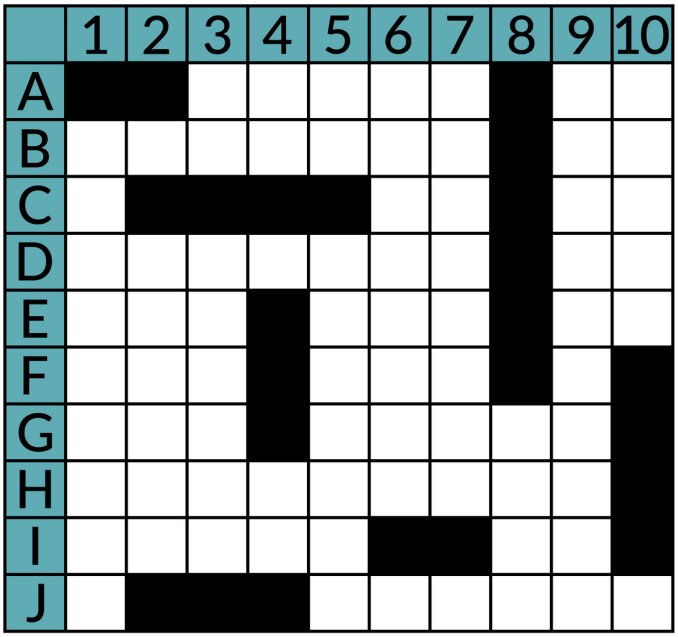
Tablice dwuwymiarowe: definiowanie macierzy i operacje na ich elementach

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



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

Wypełniliśmy już swoją planszę, zaznaczając na niej nasze okręty.



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

Część pól dalej jest pusta, część została zaczerniona (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:

```
tablica_w = [ [element11, element12, element13],
              [element21, element22, element23],
              [element31, element32, element33],
              ]
```

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:

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

```
        [0,  1,  1,  1,  1,  0,  0,  1,  0,  0 ],
        [0,  0,  0,  0,  0,  0,  0,  1,  0,  0 ],
        [0,  0,  0,  1,  0,  0,  0,  1,  0,  0 ],
        [0,  0,  0,  1,  0,  0,  0,  1,  0,  1 ],
        [0,  0,  0,  1,  0,  0,  0,  0,  0,  1 ],
        [0,  0,  0,  0,  0,  0,  0,  0,  0,  1 ],
        [0,  0,  0,  0,  0,  1,  1,  0,  0,  1 ],
        [0,  1,  1,  1,  0,  0,  0,  0,  0,  0 ]
    ]
```

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:

```
tablica_glowna = [['a', 'b', 'c'],
                  ['d', 'e', 'f'],
                  ['g', 'h', 'i']]

print(tablica_glowna[0])
# Wynik: ['a', 'b', 'c']

print(tablica_glowna[0][0])
# Wynik: 'a'

print(tablica_glowna[0][2])
# Wynik: 'c'

print(tablica_glowna[2])
# Wynik: ['g', 'h', 'i']

print(tablica_glowna[2][1])
```

```
# Wynik: 'h'

print(tablica_glowna[2][2])
# 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 Pyton 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:

```
tablica_a = [1, 2, 3]
tablica_b = [33, 23, 13, 45, 56]

print(tablica_a[2])
# Wynik: 3

# Tu przypisujemy wartość do elementu
tablica_a[2] = 999

print(tablica_a[2])
# Wynik: 999

print(tablica_b[3])
# Wynik: 45

# Tu przypisujemy wartość do elementu
tablica_b[3] = 2020
print(tablica_b[3])
# Wynik: 2020

print(tablica_b)
# 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 `poczatkowy` do `koncowy`. Przyjmiemy założenie, że liczba elementów w wierszu i w kolumnie będzie identyczna:
>
> **Specyfikacja problemu:**
>
> *Dane:*
>
> - `poczatkowy` – początek przedziału
> - `koncowy` – koniec przedziału

Wynik:

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

```
poczatkowy = 1
koncowy = 21
tablica = [ ]

# tworzymy pierwszy wiersz, bazując na generatorze listowym
wiersz = [ '*' ]  + [ x for x in range(poczatkowy, koncowy)]
dl_wiersz = len(wiersz)
liczba_elementow = koncowy - poczatkowy
tablica.append(wiersz)

# tworzymy odpowiednią liczbę wierszy (tablic jednowymiarowych)
for element in wiersz:
    if element != '*':
        nowy_wiersz = [ element ] + ( [ None ] * liczba_element
        tablica.append(nowy_wiersz)

# obliczamy wynik operacji (mnożenie) i wpisujemy do odpowiedni
# elementów tablic zagnieżdżonych
for i in range(1,dl_wiersz):
    mnoznik1 = tablica[0][i]
    for j in range(1,dl_wiersz):
        mnoznik2 = tablica[j][0]
        wynik = mnoznik1 * mnoznik2
        tablica[j][i] = wynik

# wypisujemy wyniki na ekranie
print(' ')
for wiersz in tablica:
    for element in wiersz:
        print("{:>5}|".format(element),end="")
    print(' ')

```

Rezultat działania funkcji wygląda następująco:

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

```
elementy = [ 'Python', 3.6, 'Linux', 'Windows', 'MacOS' ]
for ind , wart in enumerate(elementy):
    print('Indeks', ind, 'Wartość', wart)

Indeks 0 Wartość Python
Indeks 1 Wartość 3.6
Indeks 2 Wartość Linux
Indeks 3 Wartość Windows
Indeks 4 Wartość MacOS

elementy_dwuwymiarowe = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for ind , wart in enumerate(elementy_dwuwymiarowe):
    print('Indeks', ind, 'Wartość', wart)
```

```
Indeks 0 Wartość [1, 2, 3]
Indeks 1 Wartość [4, 5, 6]
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:

```
A = [[1, 2], [3, 4]]
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:

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

Macierz B

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

Którą w kodzie możemy zapisać jako:

```
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:

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

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

```
def suma_macierzy(A, B):
    rozmiar_macierzy = len(A)
    C = [[0 for i in range(rozmiar_macierzy)] for i in range(ro

    for x in range(rozmiar_macierzy):
        for y in range(rozmiar_macierzy):
            C[x][y] = A[x][y] + B[x][y]

    return C
```

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

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

print(suma_macierzy(A, B))
# Wynik: [[6, 8], [10, 12]]
```

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

```
def suma_macierzy(A, B):
    n = len(A)
    m = len(A[0])
    C = [[0 for i in range(m)] for i in range(n)]

    for x in range(n):
        for y in range(m):
            C[x][y] = A[x][y] + B[x][y]

    return C

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

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:

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

# przykłady wyświetlania poszczególny elementów wskazywanych po
print(lw[1])
[[5, 6], [7, 8]]

print(lw[0][1])
[3, 4]

print(lw[1][1][0])
7
```

Możemy również wykorzystać pętle, aby pokazać kolejne zagnieżdżenia:

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

for element in lw:
    print(element, type(element))
    print("Wewnątrz 1:")
    for elem in element:
        print(elem, type(elem))
        print("=> Wewnątrz 2:")
        for elem2 in elem:
            print(elem2, type(elem2))

# efekt wykonania

[[1, 2], [3, 4]] <class 'list'>
Wewnątrz 1:
[1, 2] <class 'list'>
=> Wewnątrz 2:
1 <class 'int'>
2 <class 'int'>
[3, 4] <class 'list'>
=> Wewnątrz 2:
3 <class 'int'>
4 <class 'int'>
[[5, 6], [7, 8]] <class 'list'>
Wewnątrz 1:
[5, 6] <class 'list'>
=> Wewnątrz 2:
5 <class 'int'>
6 <class 'int'>
[7, 8] <class 'list'>
=> Wewnątrz 2:
7 <class 'int'>
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 `"Przyklad"[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.

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

Ogólny zapis notacji wycinków:

```
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?

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

Polecenie 2

Przygotuj notatkę ze swoimi spostrzeżeniami dotyczącymi apletu.

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Wskaż, które z poleceń należy zastosować, aby skopiować wartości elementów jednej listy do innej.

- ◯ `nowa_lista = copy(stara_lista)`
- ◯ `nowa_lista = stara_lista[:]`
- ◯ `nowa_lista = stara_lista.copy_elements()`

Ćwiczenie 2

Wskaż, co będzie wynikiem użycia następującej komendy w języku Python: `print(['Python', 'jest'][1][2])`.

- ◯ wyświetlenie litery 'y'
- ◯ wyświetlenie litery 's'
- ◯ komunikat o błędzie: `IndexError: list index out of range`

Ćwiczenie 3

Dana jest tablica o wymiarach n × 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 × 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:

```
3 4 5 6 7
4 5 6 7 8
```

Twoje zadania

1. Program wyświetla na ekranie zawartość jedynie tych wierszy tablicy, których wszystkie elementy są większe od 2. Elementy w wierszach należy rozdzielić za pomocą spacji, każdy wiersz powinien być wydrukowany w osobnej linii.

```
n = 5
tablica = [[0 for i in range(n)] for j in range(n)]

# Tu uzupełnij kod
```

```
```

```
1
```

Ćwiczenie 4

Napisz program wypisujący kwadrat sumy wartości znajdujących się na obu przekątnych tablicy o wymiarach n × 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 × 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:

```
1600
```

Twoje zadania

1. Program wyświetla kwadrat sumy wartości leżących na przekątnych tablicy dwuwymiarowej.

```
n = 5
tablica = [[0 for i in range(n)] for j in range(n)]

# Tu uzupełnij kod
```

```
1
```

Ćwiczenie 5

Dane są dwie tablice: `macierz_a, macierz_b` o wymiarach n × 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:

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

Specyfikacja problemu:

Dane:

- `macierz_a` – tablica o wymiarach n × m
- `macierz_b` – tablica o wymiarach n × 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:

```
123 654 865 278
243 543 235 114
543 535 654 424
```

Napisz program, który wypełni tablicę `macierz_wynik` w opisany sposób, a następnie ją wypisze.

Twoje zadania

1. Program wypełnia tablicę `macierz_wynik` zgodnie z opisanym wzorem, a następnie wypisuje ją wierszami. Między poszczególnymi elementami wiersza powinien znajdować się pojedynczy znak odstępu; każdy wiersz powinien być wyświetlony w nowej linii.

```
macierz_a = [[346, 654, 865], [243, 765, 869], [543, 758, 865]]
macierz_b = [[123, 765, 867], [356, 543, 235], [649, 535, 654]]
n = 3
m = 3

macierz_wynik = [[0 for i in range(n)] for j in range(m)]

# Tu uzupełnij kod
```

```

```

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

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 5 z sekcji „Sprawdź się”.
2. 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.