UNIWERSYTET EKONOMICZNY W POZNANIU

UNIWERSYTET EKONOMICZNY W POZNANIU · KATEDRA INFORMATYKI EKONOMICZNEJ

Algorytmy i struktury danych

Zajęcia 2 · listy w Pythonie
rok akademicki 2026/2027 · dr inż. Szczepan Górtowski

Po tych zajęciach

- czytasz listę indeksami i wycinkami — także od końca
- zmieniasz listę i omijasz trzy pułapki: przesunięte indeksy, None, wspólne wiersze
- odróżniasz kopię od drugiej etykiety na tę samą listę
- wiesz, dlaczego append jest szybki, a insert(0) wolny
- używasz listy jako stosu i deque jako kolejki

Dziś: od pudełek na stole do listy pod maską

Notatnik 02_Listy_w_Pythonie.ipynb — nowa wersja z ramkami „Przewidź, zanim uruchomisz”.

01	02	03	04	05	06
Czytasz	**Kartki**	**Zmieniasz**	**Kopiujesz**	**Pod maską**	**Zadania**
indeksy, wycinki, len	Lista na stole	append, pop, sort	etykiety, copy, deepcopy	pamięć, czas, stos, kolejka	2.0–2.3 i 2.4

Ćw. 0 · Przewidź wycinki, zanim uruchomisz

5 min

1. Nie uruchamiaj komórki z sekcji 6.
2. Na kartce zapisz wynik każdej z siedmiu linii.
3. Uruchom komórkę i policz trafienia.

SEKCJA 6 NOTATNIKA — BEZ KOMENTARZY

```
lista = [2, 4, 5, 6, 9, -1, 477, 34668]

print(lista[2:6])
print(lista[-4:-1])
print(lista[:4])
print(lista[2:])
print(lista[::2])
print(lista[2:6:2])
print(lista[::-1])
```

PYTANIE

Co wypisze lista[2:6]? Czy element o indeksie 6 wchodzi do wyniku?

01

Lista i indeksy: pudełka z numerami od zera

Lista Pythona to w środku tablica — dlatego lista[i] działa od razu

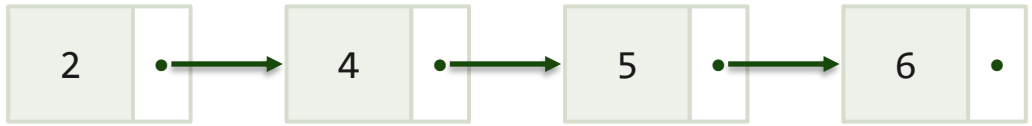
Dwa sposoby trzymania kolekcji w pamięci. Python wybrał tablicę wskaźników, która sama rośnie.

TABLICA: ELEMENTY OBOK SIEBIE



adres elementu i = początek + i × rozmiar → skok od razu, O(1)

LISTA JEDNOKIERUNKOWA: KAŻDY WSKAZUJE NASTĘPNY



do elementu i idziesz po strzałkach od początku → O(n)

LISTA W PYTHONIE

- tablica wskaźników do obiektów — stąd różne typy w jednej liście
- rezerwuje miejsce na zapas i rośnie skokami
- szybka na końcu, wolna na początku

DO ZAPAMIĘTANIA

Nazwa „lista” w Pythonie to nie lista jednokierunkowa z podręcznika algorytmów.

Python Wiki, TimeComplexity (edycja 14.02.2026): „Internally, a list is represented as an array”.

len(lista_przyklad) = 4: lista w liście to jeden element

Lista może trzymać różne typy naraz — także inną listę. len liczy tylko pierwszą warstwę.



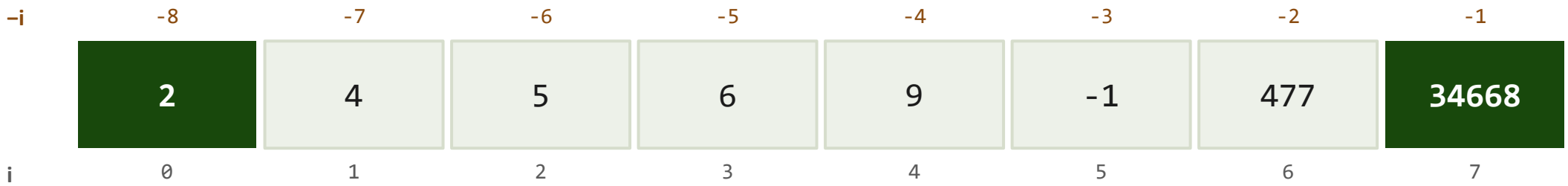

```
print(len(lista), len(lista_studentow), len(lista_przyklad))
print(len([]))
```

WYNIK

```
8 2 4
0
```

Indeks 0 to pierwszy element, −1 — ostatni

Indeksy dodatnie liczysz od początku, ujemne od końca. Poza zakresem Python podnosi IndexError.




```python
print(lista_studentow[0])
print(lista_studentow[1])
print(lista[-1])

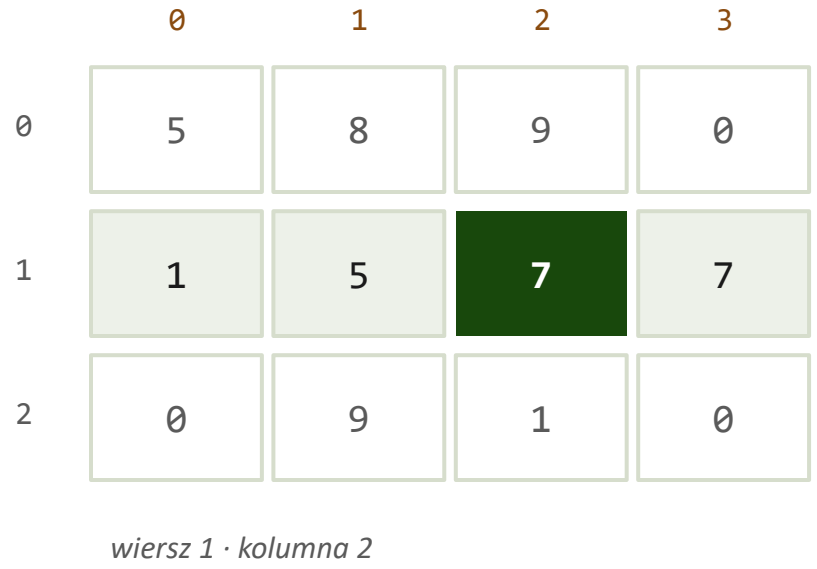
try:
    print(lista[999])
except IndexError as e:
    print("IndexError:", e)
```

WYNIK

```
Anna Nowak
Jan Kowalski
34668
IndexError: list index out of range
```

lista_macierz[1][2] = 7: najpierw wiersz, potem kolumna

Lista list to najprostsza macierz. Każda warstwa ma własne indeksy.



wiersz 1 · kolumna 2

```
print(lista_przyklad[0][2])   # w liście nr 0 element nr 2
print(lista_macierz[1][2])    # wiersz 1, kolumna 2
```

WYNIK

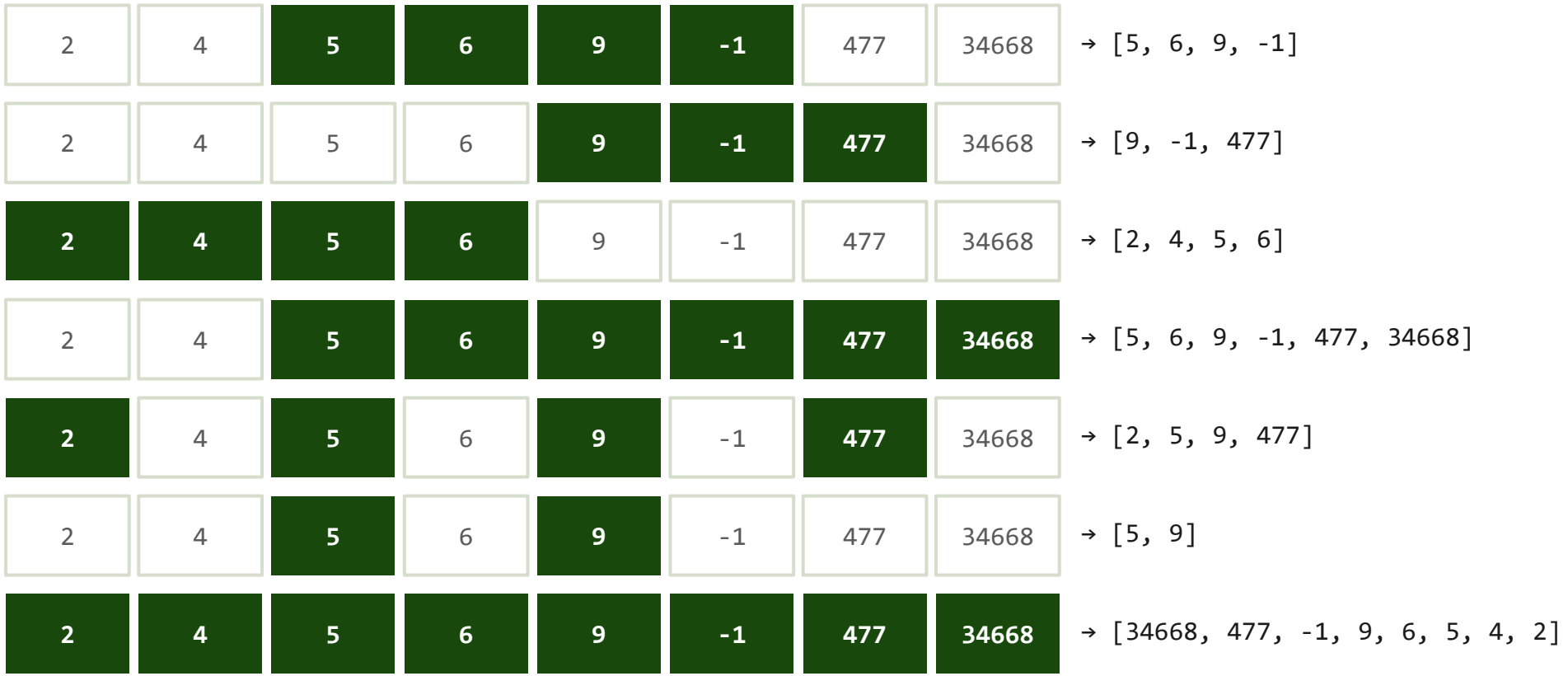
```
test
7
```

CZYTASZ OD LEWEJ

- lista_przyklad[0] → [232, 4, 'test']
- lista_przyklad[0][2] → 'test'

Klucz do ćw. 0: koniec wycinka nie wchodzi do wyniku

Wybrane elementy na zielono. lista[a:b:c] — od a, do b − 1, co c.



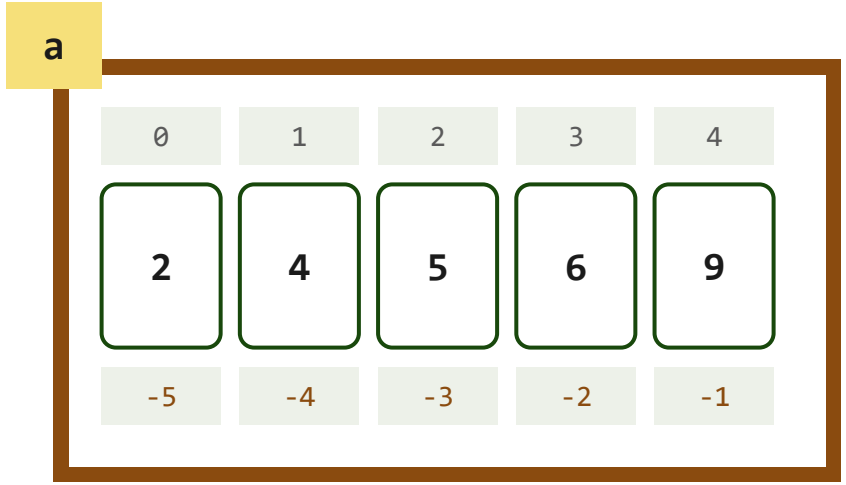
02

Ćwiczenie na kartkach: Lista na stole

Ćw. 1 · Lista na stole — bez komputera

15 min

1. Grupy 2–3 osób. Każda dostaje kartę A4 i komplet kart do wycięcia (albo karteczki samoprzylepne).
2. Ułóżcie rząd kart z listą, nad nim pasek indeksów 0–7, pod nim −8…−1.
3. Wykonujcie polecenia z karty po kolei: przesuwajcie karty i zapisujcie wynik.
4. Runda 3: karteczki a, b, c to nazwy zmiennych — przyklejacie je do rzędów kart.



PYTANIE

Ile list leży na stole po rundzie 3, a ile jest etykiet?

Klucz rund 1–2: po pop(1) pod indeksem 4 stoi już 477

Wyniki policzone w Pythonie na tej samej liście co na kartach.

Runda 1	Wynik
`lista[3]`	`6`
`lista[-1]`	`34668`
`lista[-8]`	`2`
`lista[2:6]`	`[5, 6, 9, -1]`
`lista[::2]`	`[2, 5, 9, 477]`
`lista[::-1]`	`[34668, 477, -1, 9, 6, 5, 4, 2]`

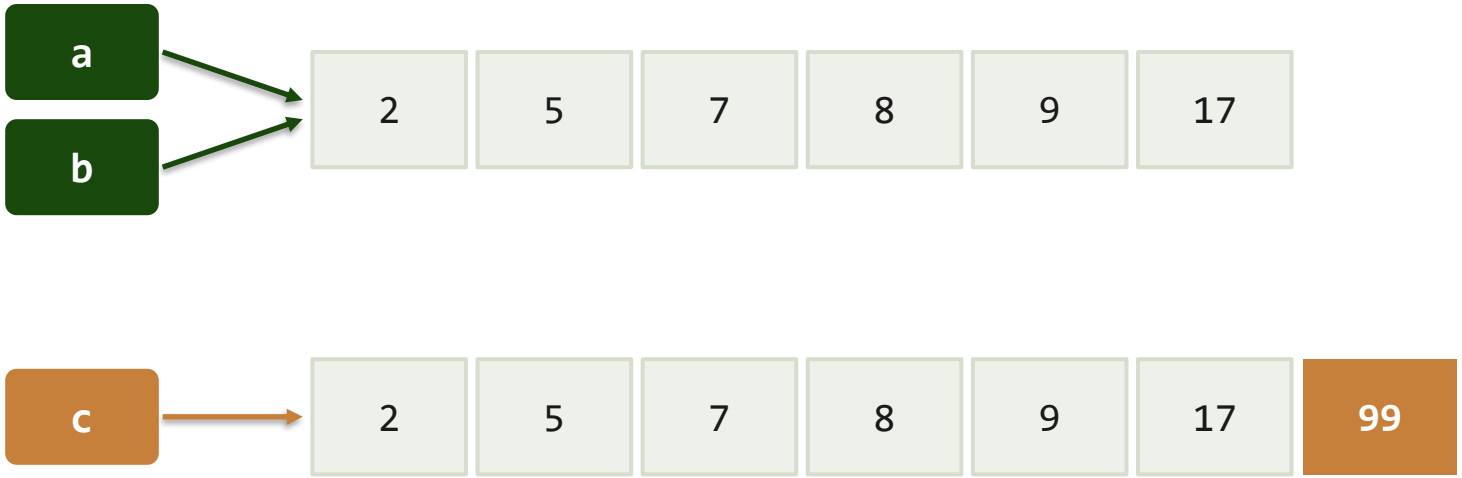
Runda 2	Zwraca	Lista po kroku
`lista.append(555)`	—	`[2, 4, 5, 6, 9, -1, 477, 34668, 555]`
`x = lista.pop()`	`555`	`[2, 4, 5, 6, 9, -1, 477, 34668]`
`lista.pop(2)`	`5`	`[2, 4, 6, 9, -1, 477, 34668]`
`lista[3] = -986`	—	`[2, 4, 6, -986, -1, 477, 34668]`

PUŁAPKA Z RUNDY 2

pop(1), potem pop(4) usuwa 4 i 477, a nie 4 i -1. Usuwaj od największego indeksu: pop(4), pop(1).

Klucz rundy 3: b = a nie kopiuje — dwie listy, trzy etykiety

b.append(17) zmienia jedyną listę, którą widzą a i b. c = a.copy() układa nowy rząd kart.



SPRAWDZONE W PYTHONIE

```
print(a) → [2, 5, 7, 8, 9, 17]
print(c) → [2, 5, 7, 8, 9, 17, 99]
b is a   → True
c is a   → False
```

ZOBACZ TO SAM

- pythontutor.com: wklej kod rundy 3
- narzędzie rysuje strzałki od nazw do list

03

Zmiany w liście i dwie pułapki

append dokłada na koniec, pop zdejmuje i oddaje element

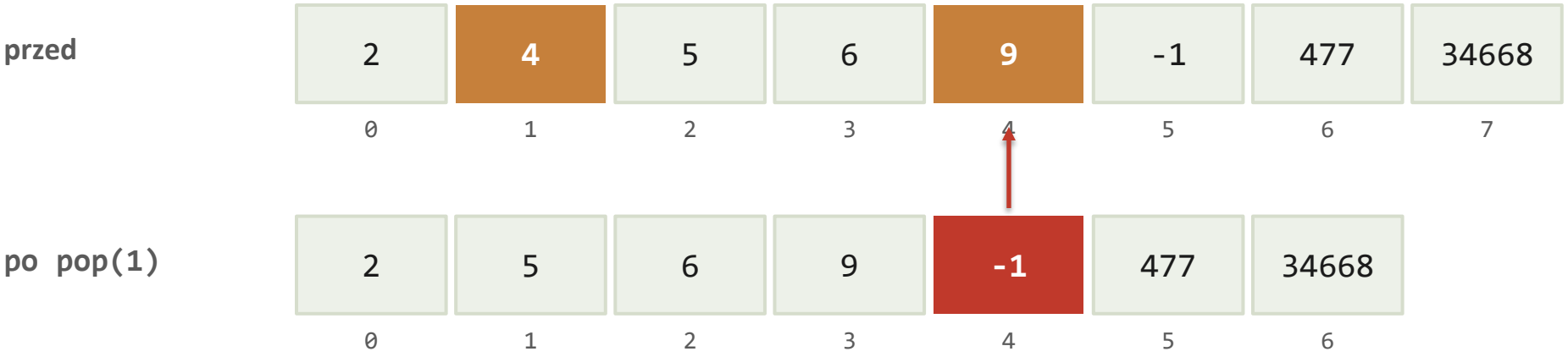
Komórka z sekcji 7, krok po kroku. Na zielono to, co właśnie się zmieniło.

`start`	2	4	5	6	9	-1	477	34668	
`append(555)`	2	4	5	6	9	-1	477	34668	**555**
`pop() → 555`	2	4	5	6	9	-1	477	34668	
`pop(2)`	2	4	6	9	-1	477	34668		
`[3] = -986`	2	4	6	**-986**	-1	477	34668		

pop(2) usunął 5 — reszta przesunęła się w lewo.

pop(1), potem pop(4) usuwa 4 i -1 — nie 9

Pułapka 1 z notatnika: po pierwszym pop wszystko na prawo przesuwa się o jedno miejsce.




```python
lista = [2, 4, 5, 6, 9, -1, 477, 34668]
print("usunięto:", lista.pop(1), "i", lista.pop(4))
print(lista)

lista = [2, 4, 5, 6, 9, -1, 477, 34668]
print("od końca:", lista.pop(4), "i", lista.pop(1))
print(lista)
```

WYNIK

```
usunięto: 4 i -1
[2, 5, 6, 9, 477, 34668]
od końca: 9 i 4
[2, 5, 6, -1, 477, 34668]
```

sort() zmienia listę i zwraca None — sorted() daje nową

Pułapka 2. Znasz ją z zajęć 1: funkcja bez return też oddaje None.

```
liczby = [5, 2, 9, 1]
wynik = liczby.sort()
print("wynik =", wynik, "| liczby =", liczby)

liczby = [5, 2, 9, 1]
nowa = sorted(liczby)
print("nowa =", nowa, "| liczby =", liczby)
```

WYNIK

```
wynik = None | liczby = [1, 2, 5, 9]
nowa = [1, 2, 5, 9] | liczby = [5, 2, 9, 1]
```

DOKUMENTACJA PYTHONA

- insert, remove i sort zwracają None
- to zasada dla wszystkich mutowalnych struktur

04

Kopia czy referencja: ile list naprawdę masz

b = a przykleja drugą etykietę — kopię robi a.copy()

Sekcja 8. Operator is sprawdza, czy dwie nazwy wskazują ten sam obiekt.

```
a = 10
b = a
print("przed: b is a ->", b is a)
b += 1
print("a =", a, "| b =", b, "| b is a ->", b is a)

a = [2, 5, 7, 8, 9]
b = a            # ta sama lista, dwie etykiety
b.append(17)
print("a =", a, "| b =", b, "| b is a ->", b is a)

a = [2, 5, 7, 8, 9]
c = a.copy()     # nowa lista z tymi samymi elementami
c.append(17)
print("a =", a, "| c =", c, "| c is a ->", c is a)
```

WYNIK

```
przed: b is a -> True
a = 10 | b = 11 | b is a -> False
a = [2, 5, 7, 8, 9, 17] | b = [2, 5, 7, 8, 9, 17] | b is a -> True
a = [2, 5, 7, 8, 9] | c = [2, 5, 7, 8, 9, 17] | c is a -> False
```

POPRAWKA DO NOTATNIKA Z 2025 R.

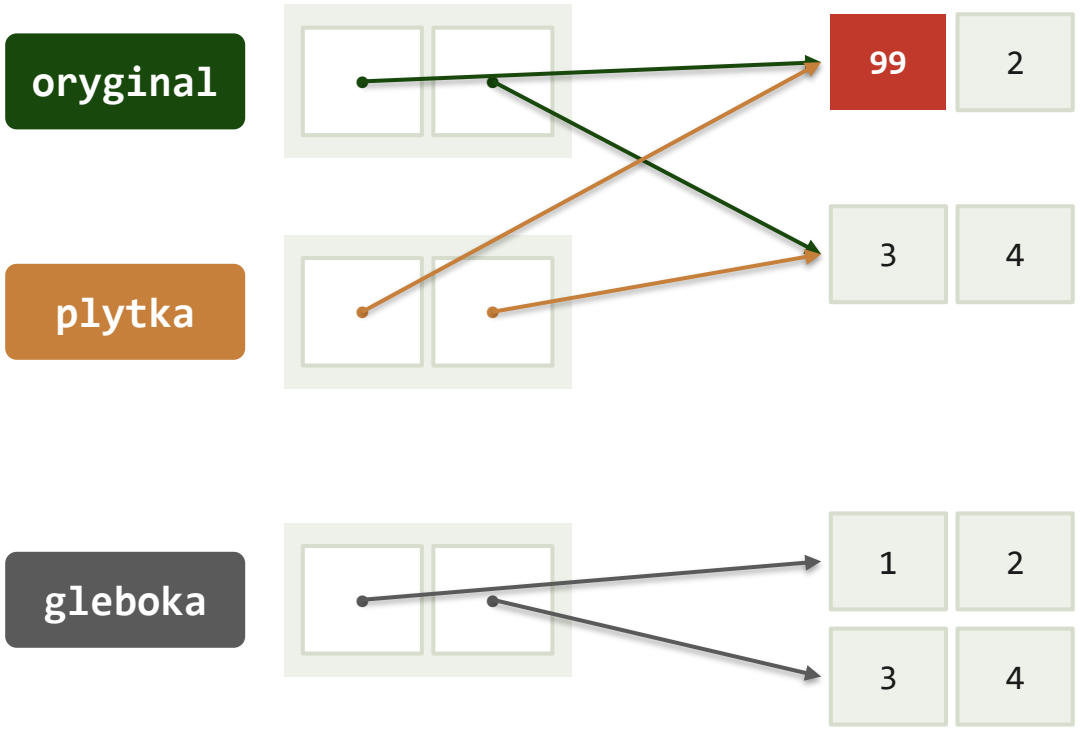
- przypisanie zawsze kopiuje referencję — także dla int
- int jest niemutowalny: b += 1 przykleja b do nowej liczby
- lista jest mutowalna: b.append(17) zmienia wspólny obiekt

TEST W JEDNEJ LINII

b is a → True znaczy: to ta sama lista. Zmiana przez b będzie widoczna w a.

copy() kopiuje tylko pierwszą warstwę: wewnętrzne listy są wspólne

Kopia płytka i głęboka. plytka[0][0] = 99 zmienia też oryginał.




```python
import copy

oryginal = [[1, 2], [3, 4]]
plytka = oryginal.copy()
gleboka = copy.deepcopy(oryginal)

plytka[0][0] = 99
print("oryginal:", oryginal)
print("plytka:  ", plytka)
print("gleboka: ", gleboka)
```

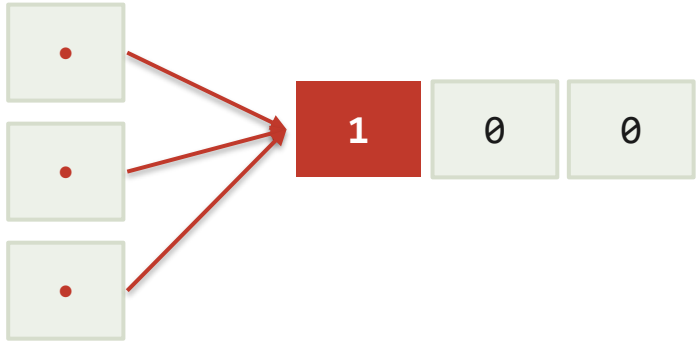
WYNIK

```
oryginal: [[99, 2], [3, 4]]
plytka:   [[99, 2], [3, 4]]
gleboka:  [[1, 2], [3, 4]]
```

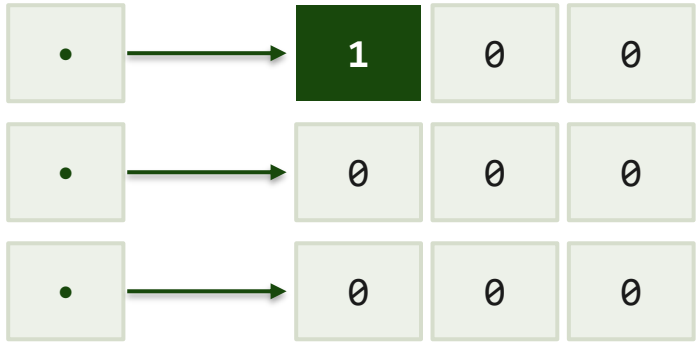
[[0] * 3] * 3 to trzy etykiety na jeden wiersz

Pułapka 3. Mnożenie listy powiela referencje, a nie wiersze.

ZLA = [[0] * 3] * 3



DOBRA — KAŻDY WIERSZ OSOBNO




```
zla = [[0] * 3] * 3
zla[0][0] = 1
print("zła:   ", zla)

dobra = []
for _ in range(3):
    dobra.append([0] * 3)
dobra[0][0] = 1
print("dobra: ", dobra)
```

WYNIK

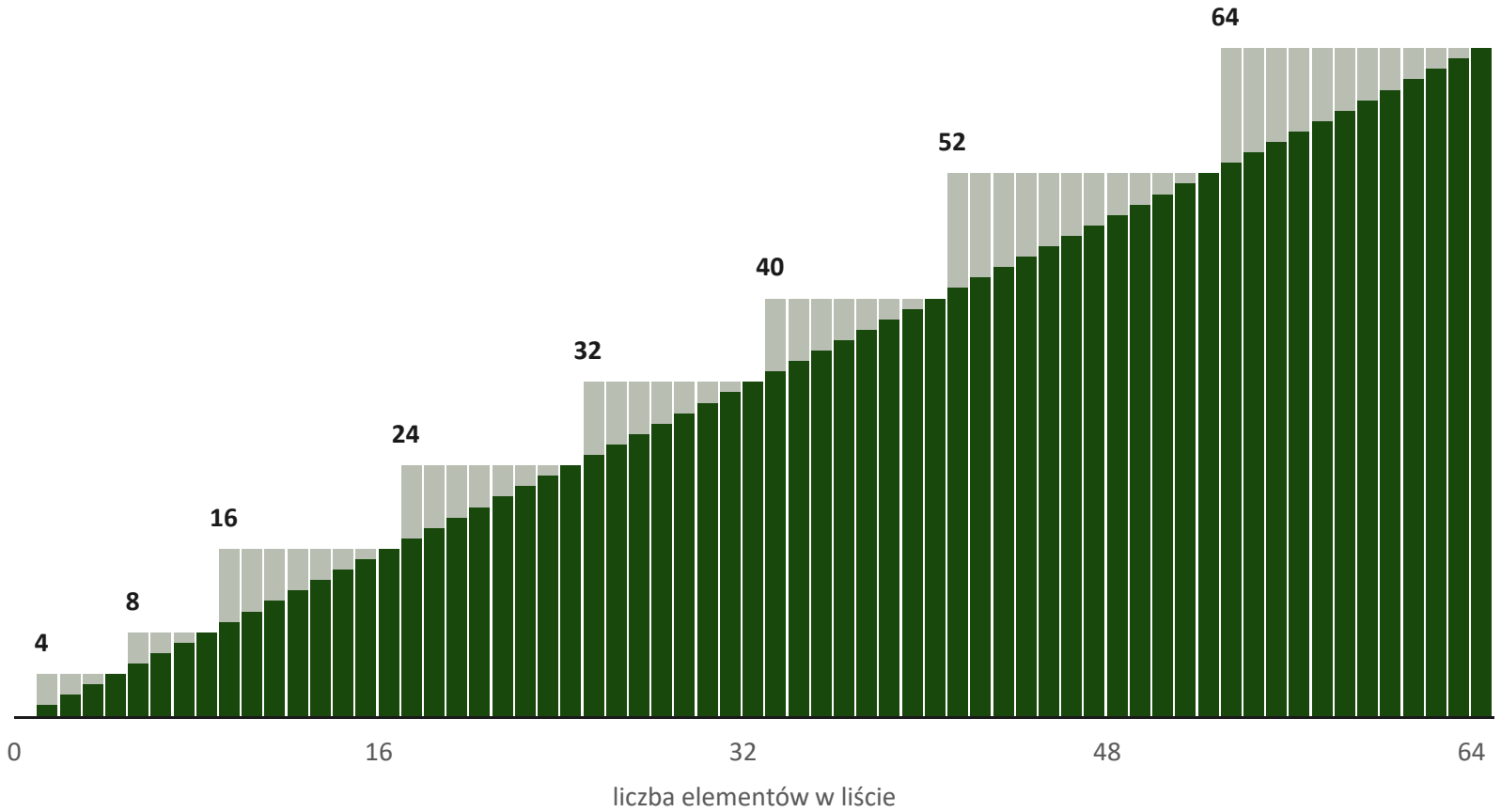
```
zła:    [[1, 0, 0], [1, 0, 0], [1, 0, 0]]
dobra:  [[1, 0, 0], [0, 0, 0], [0, 0, 0]]
```

05

Lista pod maską: pamięć, czas, stos i kolejka

Lista rośnie skokami: przy 9. elemencie ma już 16 miejsc

Pomiar sys.getsizeof z sekcji 9 notatnika. Szare — miejsca zarezerwowane, zielone — zajęte.



SKOKI POJEMNOŚCI

```
 1 el. →  4
 5 el. →  8
 9 el. → 16
17 el. → 24
25 el. → 32
33 el. → 40
41 el. → 52
53 el. → 64
```

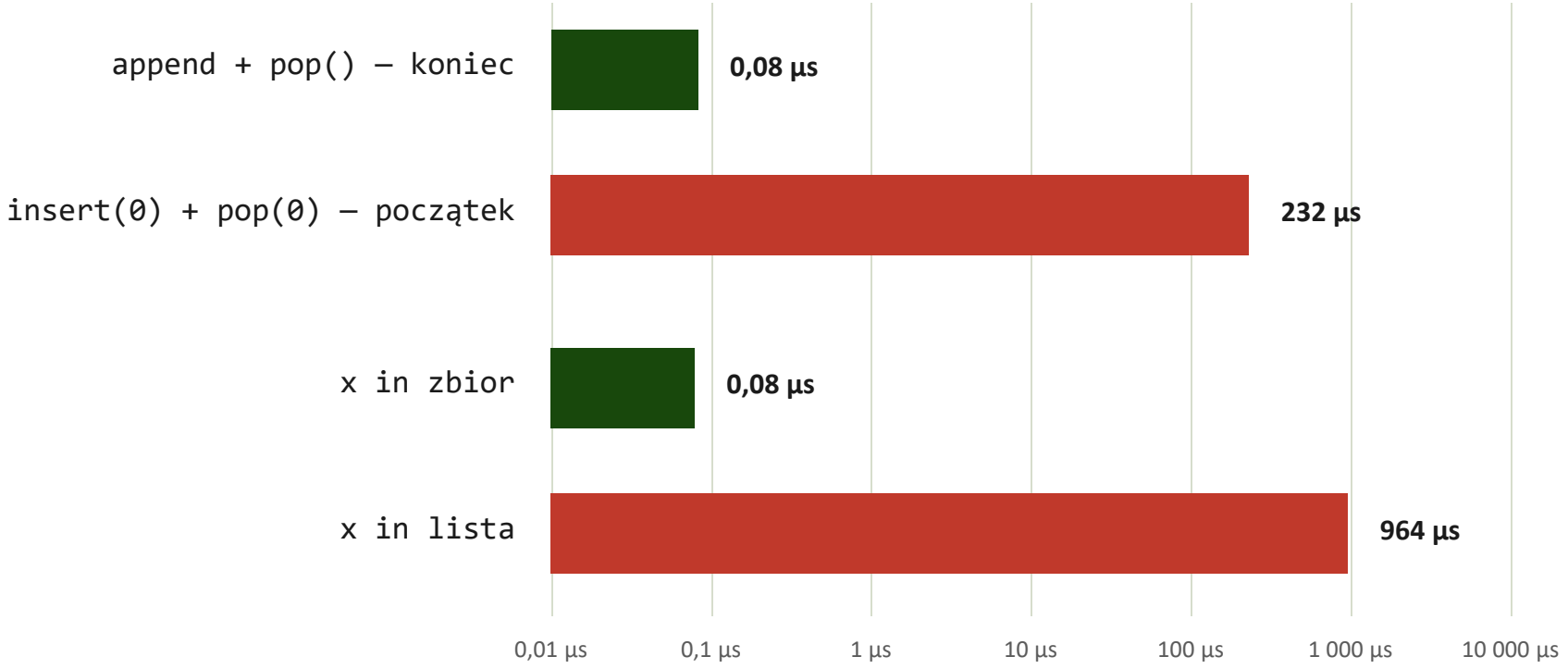
DLATEGO

append jest średnio O(1): przeprowadzka zdarza się rzadko.

Pomiar: Python 3.11.9 (Windows); każde miejsce to wskaźnik 8 B. W Colab liczby mogą się różnić.

Wstawienie na początek jest ok. 2 800 razy wolniejsze niż na koniec

Pomiar timeit z sekcji 9 na liście 100 000 liczb. Oś pozioma logarytmiczna: każda kreska to ×10.



DLACZEGO

- początek: przesuwasz 100 000 wskaźników
- in: przeglądasz po kolei

WNIOSEK

Szukasz często? Zbiór: ok. 12 000× szybciej.

Pomiar: Python 3.11.9 (Windows), minimum z 5 powtórzeń. Na innym komputerze wyjdą inne liczby, proporcje podobne.

O(1) na końcu, O(n) na początku i przy szukaniu

Złożoność średnia operacji na liście w CPython. n — długość listy, k — długość wycinka.

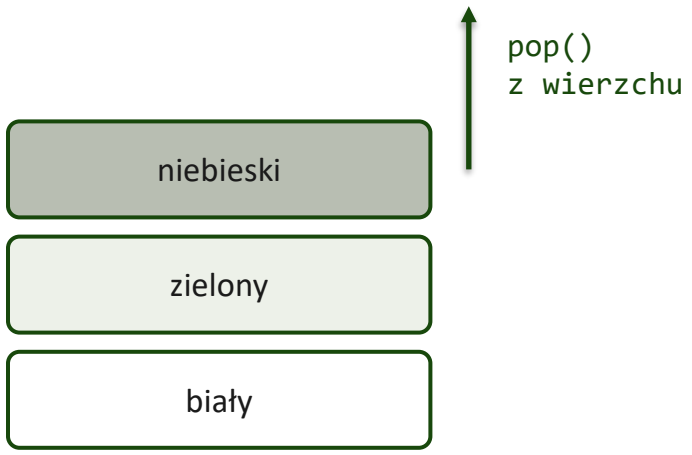
Operacja	Złożoność	Bo
`lista[i], lista[i] = x`	`O(1)`	adres liczony od razu
`len(lista)`	`O(1)`	lista pamięta swoją długość
`append(x), pop()`	`O(1)`	koniec — nic się nie przesuwa
`insert(0, x), pop(0)`	`O(n)`	przesuwa wszystkie elementy
`x in lista`	`O(n)`	sprawdza element po elemencie
`lista.copy(), lista[a:b]`	`O(n) / O(k)`	kopiuje elementy
`x in zbior`	`O(1)`	zbiór haszuje elementy

Python Wiki, TimeComplexity, wiki.python.org/moin/TimeComplexity (edycja 14.02.2026, sprawdzone 8.10.2026).

Lista to dobry stos, ale zła kolejka — do kolejki weź deque

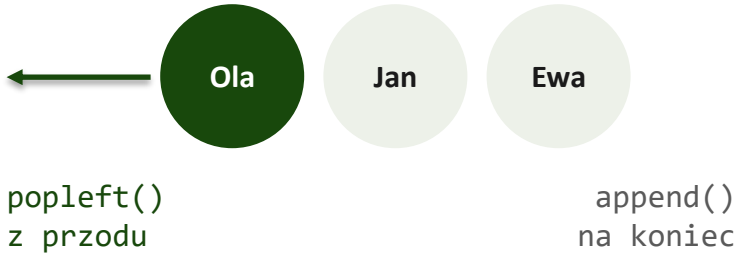
Stos: ostatni przyszedł, pierwszy wychodzi (LIFO). Kolejka: pierwszy przyszedł, pierwszy wychodzi (FIFO).

STOS: LISTA



append() kładzie na wierzch

KOLEJKA: DEQUE



WYNIK

```
zdejmuję ze stosu: niebieski
obsługuję z kolejki: Ola
```

DOKUMENTACJA PYTHONA

- lista jako kolejka jest nieefektywna: pop(0) przesuwa resztę
- do kolejki: collections.deque — szybka z obu końców

docs.python.org/3/tutorial/datastructures.html, 5.1.1–5.1.2 (sprawdzone 8.10.2026). Kod: sekcja 10 notatnika.

Od Pythona 3.11 lista sortuje się regułą z Powersorta

Ciekawostka z algorytmiki: wynik badań z 2018 r. trafił do Pythona w 2022 r.

2002

Tim Peters pisze Timsort

w Pythonie od wersji 2.3

2018

Powersort

J. I. Munro, S. Wild, konferencja ESA

2022

Python 3.11

reguła scalania z Powersorta zastępuje tę z Timsorta

SORTOWANIE JEST STABILNE

Ola, Jan i Ewa mają po 3 litery i przy key=len zostają w kolejności z wejścia.

```
studenci = ["Ola", "Bartłomiej", "Jan", "Ewa", "Zuzanna"]
print(sorted(studenci))
print(sorted(studenci, key=len))
print(sorted(studenci, key=len, reverse=True))
```

WYNIK

```
['Bartłomiej', 'Ewa', 'Jan', 'Ola', 'Zuzanna']
['Ola', 'Jan', 'Ewa', 'Zuzanna', 'Bartłomiej']
['Bartłomiej', 'Zuzanna', 'Ola', 'Jan', 'Ewa']
```

University of Liverpool News, 12.12.2022; Wikipedia, „Powersort” (sprawdzone 8.10.2026).

06

Pętle i warunki: przechodzisz po liście

range(5) daje 0–4, range(2, 11, 2) — liczby parzyste do 10

range(start, stop, krok) — jak wycinek: stop nie wchodzi. for przechodzi po elementach, while — dopóki warunek jest prawdziwy.




```python
for i, student in enumerate(["Anna Nowak", "Jan Kowalski"]):
    print(i, student)

print(list(range(2, 11, 2)))
```

WYNIK

```
0 Anna Nowak
1 Jan Kowalski
[2, 4, 6, 8, 10]
```

KIEDY CO

- for — znasz kolekcję albo liczbę powtórzeń
- while — powtarzasz, dopóki coś się nie zmieni
- enumerate — potrzebujesz indeksu i elementu

ocena 3 and obecność 0.7 → „zaliczono”: oba warunki prawdziwe

Sekcja 13: if, elif, else i operatory logiczne.

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

```
ocena_test = 3
obecnosc = 0.7
if ocena_test >= 3 and obecnosc > 0.5:
    print("zaliczono")
else:
    print("niezaliczono")
```

WYNIK

```
zaliczono
```

07

Zadania 2.0–2.4: test powie, czy działa

Ćw. 2 · Zadanie 2.0 i 2.1: operacje i kopie

8 min

1. Zadanie 2.0: lista a z co najmniej 10 elementami, len, indeksy 3, 7, −1, −8, wycinki a[2:6] i a[::3].
2. Dodaj 4 elementy, usuń indeksy 2 i 8 — w dobrej kolejności.
3. Zadanie 2.1: b = a i b.append(...), potem c = a.copy() i c.append(...). Sprawdź b is a i c is a.

PYTANIE

W jakiej kolejności usuniesz indeksy 2 i 8? Dlaczego?

Ćw. 3 · Zadanie 2.2 i 2.3: pętle, warunki, test

7 min

1. Zadanie 2.2: liczby 1–10 w while, parzyste w for z range, math.pi z 5 miejscami po przecinku.
2. Zadanie 2.3: uzupełnij parzyste_zakres(a, b) w sekcji 16.
3. Uruchom komórkę z funkcją, potem komórkę Sprawdź.

SEKCJA 16 — SZABLON (SKRÓCONY)

```
def parzyste_zakres(a: int, b: int) -> list:
    """Zwróć listę liczb parzystych z domkniętег…"""
    Jeśli a > b, zwróć pustą listę.
    """
    # TODO: zaimplementuj

    return
```

PYTANIE

Co zwróci parzyste_zakres(10, 1)?

Rezultat zadania 2.3: pętla, if i append — 4 testy zaliczone

Przykładowe rozwiązanie z range i if, bez składni, której jeszcze nie znamy.

```python
def parzyste_zakres(a: int, b: int) -> list:
    """Zwróć listę liczb parzystych z domkniętego zakresu [a, b]."""
    wynik = []
    for x in range(a, b + 1):
        if x % 2 == 0:
            wynik.append(x)
    return wynik
```

KOMÓRKA SPRAWDŹ

```
✅ Wszystkie testy zaliczone!
```

SPRAWDŹ U SIEBIE

- parzyste_zakres(1, 10): [2, 4, 6, 8, 10]
- parzyste_zakres(3, 3): []
- parzyste_zakres(10, 1): []

Ćw. 4 · Zadanie 2.4 dla chętnych: unikalne elementy

dom

1. Utwórz pustą listę wynik.
2. Przejdź po liście; element dokładaj, jeśli go jeszcze nie ma w wyniku.
3. Nie zmieniaj listy, którą dostała funkcja.

SEKCJA 16 — SZABLON (SKRÓCONY)

```python
def unikalne(lista: list) -> list:
    """Elementy bez powtórzeń, w kolejności pier…"""
    # TODO: dokładaj do wyniku tylko elementy, któr…

    return
```

PYTANIE

Ile porównań wykona funkcja dla 100 000 różnych liczb?

Rezultat zadania 2.4: pętla, in i append — kolejność zostaje

Przykładowe rozwiązanie. Działa, ale dla długich list jest wolne: in na liście to O(n).

```python
def unikalne(lista: list) -> list:
    """Elementy bez powtórzeń, w kolejności pierwszego wystąpienia."""
    wynik = []
    for x in lista:
        if x not in wynik:
            wynik.append(x)
    return wynik
```

KOMÓRKA SPRAWDŹ

```
✅ Wszystkie testy zaliczone!
```

SPRAWDŹ U SIEBIE

- unikalne([3, 1, 3, 2, 1]): [3, 1, 2]
- unikalne(["a", "a"]): ['a']
- unikalne([]): []

Na następne zajęcia

- Dokończ zadania 2.0–2.3; komórki Sprawdź mają pokazać ✅
- Odpowiedz na pytania z sekcji 17: mutowalność, referencja i kopia, szybkie i wolne operacje
- Wklej kod z sekcji 8 do pythontutor.com i obejrzyj strzałki
- Dla chętnych: zadanie 2.4 — unikalne elementy

Trzy rzeczy do zapamiętania

[a:b]

koniec wycinka nie wchodzi do wyniku

b = a

to druga etykieta, nie kopia — kopia to a.copy()

~2 800×

tyle wolniej wstawisz na początek niż na koniec

DZIĘKUJĘ

dr inż. Szczepan Górtowski

Katedra Informatyki Ekonomicznej · Uniwersytet Ekonomiczny w Poznaniu

szczepan.gortowski@ue.poznan.pl · dyżury: kie.ue.poznan.pl/pl/dyzury/