Pierwsza Ławka · zestaw działowy · liceum i technikum

Dział 11

Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium)

Dobór struktur danych (lista, kolejka deque, zbiór, słownik, Counter, kopiec heapq) i bibliotek Pythona, praca w IDE z debugerem i testami, projekt zespołowy w repozytorium Git (gałęzie, scalanie, konflikty). Osiem kart: teoria z przykładami, ćwiczenia, utrwalenie w formule CKE, zadanie przy komputerze z plikami danych (ZIP), kartkówka i sprawdzian w grupach A i B – z kluczem i zasadami oceniania.

Informatyka | klasa 3 LO i technikum | Dział 11 | zakres rozszerzony (ZR)

	Karta	Rodzaj	Poziom	Zadania
1	**Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – najważniejsze wiadomości** Teoria, przykłady krok po kroku, rozgrzewka	Najważniejsze wiadomości	ZR	4 zad. · 8 pkt
2	**Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – ćwiczenia podstawowe** Ćwiczę umiejętności z działu	Ćwiczenia	ZR	6 zad. · 19 pkt
3	**Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – utrwalenie** Zadania w formule CKE: polecenia i punktacja jak w arkuszach maturalnych	Zadania typu maturalnego	ZR	6 zad. · 11 pkt
4	**Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – w praktyce, przy komputerze** Pliki danych w archiwum ZIP · wyniki w plikach wynikiN.txt	Ćwiczenia	ZR	1 zad. · 7 pkt
5	**Kartkówka: struktury danych i Git – grupa A** 15 minut · 7 pkt · bez komputera	Kartkówka	ZR	4 zad. · 7 pkt 15 min
6	**Kartkówka: struktury danych i Git – grupa B** 15 minut · 7 pkt · bez komputera	Kartkówka	ZR	4 zad. · 7 pkt 15 min
7	**Sprawdzian: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – grupa A** 45 minut · 21 pkt · bez komputera	Sprawdzian	ZR	8 zad. · 21 pkt 45 min

Podgląd: Pierwsza Ławka

8 **Sprawdzian: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – grupa B**
45 minut · 21 pkt · bez komputera

Sprawdzian | ZR | 8 zad. · 21 pkt · 45 min

Jak korzystać z zestawu

- Karta 1 przypomina najważniejsze wiadomości i pokazuje przykłady rozwiązane krok po kroku (z kodem w Pythonie albo opisem kroków w aplikacji) – od niej warto zacząć.
- Karta 2 to ćwiczenia podstawowe, karta 3 – utrwalenie w formule zadań maturalnych CKE (ramka „Jak to sprawdza CKE").
- Karta 4 to praca przy komputerze: pliki danych są w archiwum ZIP dołączonym do zestawu, a odpowiedzi kontrolne dla plików *_przyklad (pliki o nazwie z dopiskiem „_przyklad") podano przy zadaniach. Wyniki zapisuje się w plikach wynikiN.txt – jak na maturze.
- Karty 5–6 to kartkówka (15 minut), karty 7–8 – sprawdzian (45 minut), obie w grupach A i B: te same typy zadań i ta sama punktacja, inne dane. Kartkówka i sprawdzian nie wymagają komputera.
- Klucz zawiera odpowiedzi, przykładowe programy, zasady oceniania w stylu CKE i numery wymagań podstawy. Osobny plik „Powtórka" zbiera teorię, typowe błędy i test „Sprawdź się".

Oznaczenia

ZP	zakres podstawowy
ZR	zakres rozszerzony (treści tylko dla ZR)
(0–2 pkt)	liczba punktów możliwych do zdobycia za zadanie
Zadanie 4.1.	wiązka zadań do wspólnego materiału źródłowego
Jak to sprawdza CKE	typ zadania i typowy błąd ze sprawozdań CKE
Na maturze	wymagania z podstawy programowej, których dotyczy temat

Wymagania z podstawy programowej (Dz.U. 2024 poz. 1019)

- **II.R1** projektuje i tworzy 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;
- **II.R2** sprawnie posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;
- **IV.R1** przy realizacji zespołowego projektu programistycznego posługuje się środowiskiem przeznaczonym do współpracy i realizacji projektów zespołowych;
- **I.R4** do realizacji rozwiązania problemu dobiera odpowiednią metodę lub technikę algorytmiczną i struktury danych;

Materiał oryginalny – żadne zadanie nie jest kopią zadania CKE ani podręcznika; dane w plikach są fikcyjne. Podstawa programowa LO i technikum w brzmieniu z 2024 r. (Dz.U. 2024 poz. 1019). Wymagania: II.R1, II.R2, IV.R1, I.R4. Miejsce w programach nauczania: „Informatyka na czasie" (Nowa Era) ZR – programowanie i projekty zespołowe (nazwy serii podano tylko informacyjnie). Do użytku własnego i pracy z własnymi uczniami.

Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – najważniejsze wiadomości

Teoria, przykłady krok po kroku, rozgrzewka

ZR </> KARTA 1/8

Imię i nazwisko: ..... Klasa: ..... Data: ..... Wynik: ..... / 8 pkt

Najważniejsze wiadomości · 8 pkt

Przypomnij sobie

- **Dobór struktury danych (I.R4)** zaczynamy od pytania, jakie operacje będą wykonywane najczęściej: dopisywanie na końcu, pobieranie z początku, sprawdzanie „czy jest", zliczanie, pobieranie najmniejszego. Ta sama lista operacji w złej strukturze może działać setki razy wolniej.
- **Lista (list)** – ciąg elementów z dostępem po indeksie w stałym czasie; append na końcu – szybko. Usuwanie i wstawianie na początku (pop(0), insert(0, x)) wymaga przesunięcia wszystkich elementów – czas proporcjonalny do n. Sprawdzenie x in lista przegląda elementy po kolei.
- **Kolejka dwustronna (collections.deque)** – struktura dynamiczna: dopisywanie i usuwanie z obu końców (append, appendleft, pop, popleft) w czasie stałym, O(1). Dobra na kolejkę FIFO (pierwszy przyszedł – pierwszy obsłużony) i na stos.
- **Słownik (dict) i zbiór (set)** – sprawdzanie, czy klucz występuje, oraz odczyt wartości dla klucza działają średnio w czasie stałym (tablica z haszowaniem). **Counter** to słownik do zliczania (brakujący klucz daje 0, most_common(k) – k najczęstszych), **defaultdict(list)** sam tworzy pustą listę dla nowego klucza.
- **Kopiec (moduł heapq)** – lista, w której heap[0] jest zawsze najmniejszy. heappush dodaje element, heappop zabiera najmniejszy (obie operacje w czasie rzędu log n); heapify zamienia listę w kopiec w czasie liniowym. Pary (priorytet, nazwa) porównywane są najpierw po priorytecie, a przy równym – po nazwie.
- **Sortowanie z biblioteki**: sorted(dane) tworzy nową listę, dane.sort() sortuje w miejscu i zwraca None. Parametr key= podaje funkcję, według której wyniku porównujemy, a reverse=True odwraca porządek. Sortowanie w Pythonie jest **stabilne** – elementy o równym kluczu zachowują wcześniejszą kolejność.
- **Zintegrowane środowisko programistyczne (IDE, II.R2)**, np. IDLE lub PyCharm (wymienione w komunikacie CKE o egzaminie 2027): edytor z podświetlaniem składni, uruchamianie, komunikaty o błędach z numerem wiersza i **debuger**. Punkt przerwania (breakpoint) zatrzymuje program; potem: krok z wejściem do funkcji (step, w pdb: s), krok bez wchodzenia (next, n), dalej do następnego punktu (continue, c), podgląd zmiennej (p x). Na egzaminie wtyczki AI generujące kod są wyłączone.
- **Testowanie**: instrukcja assert warunek przerywa program błędem AssertionError, gdy warunek jest fałszywy. Testy piszemy dla przykładu z treści i przypadków brzegowych (jeden element, wartości równe, pusty plik).
- **Projekt zespołowy (IV.R1)** prowadzi się w systemie kontroli wersji, np. Git, z repozytorium we wspólnej usłudze. Plik bywa *zmodyfikowany* (modified), *w poczekalni* (staged – po git add) albo *zatwierdzony* (committed – po git commit, zapisany w historii). Każdy pracuje na własnej **gałęzi** (git switch -c nazwa), potem gałęzie się scala (git merge). Gdy dwie osoby zmieniły ten sam fragment pliku, powstaje **konflikt** – Git oznacza oba warianty znacznikami, a osoba scalająca wybiera właściwą treść, usuwa znaczniki i zatwierdza wynik (git add, git commit).

Przykłady z rozwiązaniem

Przykład 1. **Ornitolodzy z koła przyrodniczego zapisali obserwowane ptaki. Policz wystąpienia gatunków biblioteką collections.**

```
from collections import Counter

gatunki = "sikora wróbel sikora kos wróbel sikora dzięcioł".split()
licz = Counter(gatunki)
print(licz["sikora"], licz["szpak"])
print(licz.most_common(2))
```

Wynik:

```
3 0
[('sikora', 3), ('wróbel', 2)]
```

Counter dla brakującego klucza („szpak”) zwraca 0 zamiast błędu KeyError; most_common(2) daje dwa najczęstsze gatunki. Bez biblioteki: dla każdego g wykonujemy d[g] = d.get(g, 0) + 1.

Przykład 2. **Zasymuluj kolejkę do szatni: „+imię” – osoba dołącza, „-” – obsługa pierwszej osoby. Dlaczego deque, a nie lista?**

```
from collections import deque

kolejka = deque()
for z in ["+Ala", "+Bartek", "-", "+Celina", "-", "+Darek"]:
    if z[0] == "+":
        kolejka.append(z[1:])        # dołącza na końcu
    else:
        print("obsłużony:", kolejka.popleft())   # obsługa z początku
print(list(kolejka))
```

Wynik:

```
obsłużony: Ala
obsłużony: Bartek
['Celina', 'Darek']
```

popleft() działa w czasie stałym. Lista z pop(0) przesuwa przy każdej obsłudze wszystkie pozostałe elementy – przy 100 000 osób w kolejce to ogromna różnica.

Przykład 3. **Zgłoszenia awarii mają priorytet (1 – najpilniejsze). W jakiej kolejności zostaną obsłużone?**

```
import heapq

zgloszenia = []
heapq.heappush(zgloszenia, (3, "drukarka"))
heapq.heappush(zgloszenia, (1, "serwer"))
heapq.heappush(zgloszenia, (2, "projektor"))
heapq.heappush(zgloszenia, (1, "router"))
while zgloszenia:
    print(heapq.heappop(zgloszenia))
```

Wynik:

```
(1, 'router')
(1, 'serwer')
(2, 'projektor')
(3, 'drukarka')
```

heappop zawsze zabiera najmniejszą parę. Dwie pary z priorytetem 1 porównują się po nazwie: „router” < „serwer”.

Przykład 4. **Uczeń pisze funkcję srednia, testuje ją w IDE i zapisuje pracę w repozytorium zespołu na osobnej gałęzi.**

```
def srednia(oceny):
    s = 0
    for x in oceny:
        s = s + x
    return s / len(oceny)

assert srednia([4, 5, 6]) == 5
assert srednia([2]) == 2
assert srednia([3, 4]) == 3.5
```

Gdyby test nie przeszedł, w IDE stawiamy punkt przerwania w wierszu s = s + x i krokami (next) obserwujemy wartości s i x. Polecenia Git:

```
git switch -c wyniki          # nowa gałąź „wyniki” i przejście na nią
git status                    # co zmienione, co w poczekalni
git add raport.py             # plik do poczekalni (staged)
git commit -m "Liczenie średniej"
git switch main
git merge wyniki              # scalenie gałęzi „wyniki” z main
```

Jeśli na main ktoś w międzyczasie zmienił ten sam wiersz, git merge zgłosi konflikt:

```
<<<<<<< HEAD
PROG = 30
=======
PROG = 25
>>>>>>> wyniki
```

Zespół ustala wartość (np. PROG = 25), usuwa trzy wiersze znaczników, potem git add i git commit.

Zadania

Zadanie 1. (0–1 pkt) ..... / 1

Która struktura z biblioteki standardowej Pythona pozwala w czasie stałym usuwać elementy z początku?

A. tuple B. list C. collections.deque D. str

Zadanie 2. (0–2 pkt) ..... / 2

Uzupełnij.

1. Funkcja heapq.heappop zwraca element _____ (najmniejszy/największy).
2. Po poleceniu git add plik jest w stanie _____ (zmodyfikowany/w poczekalni/zatwierdzony).

Zadanie 3. (0–2 pkt) ..... / 2

Oceń prawdziwość zdań.

1. Metoda lista.sort() zwraca posortowaną listę.	P	F
2. Instrukcja assert 2 + 2 == 5 przerwie program błędem AssertionError.	P	F

Zadanie 4. (0–3 pkt) ..... / 3

Dopasuj polecenie debugera pdb do jego działania. Jedno działanie jest zbędne.

1. n	a) wykonuje bieżący wiersz i wchodzi do wywoływanej funkcji
2. s	b) wykonuje bieżący wiersz bez wchodzenia do funkcji
3. c	c) uruchamia program dalej do następnego punktu przerwania
	d) usuwa wszystkie zmienne programu

1 ____ 2 ____ 3 ____

PIERWSZA ŁAWKA

Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – ćwiczenia podstawowe

Ćwiczę umiejętności z działu

ZR </> KARTA 2/8

Imię i nazwisko: ..... Klasa: ..... Data: ..... Wynik: ..... / 19 pkt

Ćwiczenia · 19 pkt

Zadanie 1. (0–3 pkt) ..... / 3

Co wypisze program? Podaj wszystkie trzy wartości.

```
d = {}
for s in ["kot", "pies", "kot", "ryba", "kot", "pies"]:
    d[s] = d.get(s, 0) + 1
print(len(d), d["kot"], sorted(d, key=lambda k: d[k]))
```

Miejsce na odpowiedź

Zadanie 2. (0–4 pkt) ..... / 4

Dopasuj problem do najlepiej pasującej struktury danych. Jedna struktura jest zbędna.

1. obsługa klientów w kolejności przyjścia	a) lista (list)
2. sprawdzanie, czy login jest już zajęty (100 000 loginów)	b) kopiec (heapq)
3. wybieranie zawsze zadania o najmniejszym terminie	c) zbiór (set)
4. odczyt i-tego pomiaru z serii	d) kolejka (deque)
	e) napis (str)

1 ____ 2 ____ 3 ____ 4 ____

Zadanie 3. (0–2 pkt) ..... / 2

Program grupuje uczniów według klas. Uzupełnij lukę i podaj, co wypisze program.

```
from collections import defaultdict

klasy = defaultdict(list)
for imie, klasa in [("Ola", "3a"), ("Igor", "3b"), ("Ewa", "3a")]:
    klasy[klasa].___(imie)        # luka 1
print(klasy["3a"], len(klasy["3c"]), len(klasy))
```

1. luka 1: _____
2. wypisany wiersz: _____

Zadanie 4. (0–2 pkt) ..... / 2

Liczby dodaje się do kopca heapq (heappush), a „pop” oznacza heappop. Wpisz w tabeli kolejne zwrócone wartości.

operacje	zwracane przez kolejne pop
7, 3, 9, pop, 1, 5, pop, pop, 2, pop	

Zadanie 5. (0–3 pkt) ..... / 3

Funkcja najwieksza(liczby) ma zwracać największy element niepustej listy. Napisz test assert, który wykryje błąd, wyjaśnij przyczynę i popraw funkcję.

```
def najwieksza(liczby):
    m = 0
    for x in liczby:
        if x > m:
            m = x
    return m
```

Miejsce na odpowiedź

Zadanie 6. (0–5 pkt) ..... / 5

Ustaw polecenia Git w kolejności pracy nad zmianą na osobnej gałęzi: git add, git commit, git merge poprawka, git switch -c poprawka, git switch main.

1. _____
2. _____
3. _____
4. _____
5. _____

Zadanie 4. (0–2 pkt) ..... / 2

Przeanalizuj program. Uzupełnij tabelę – podaj wartości wypisane przez program.

```
import heapq
h = [8, 2, 6]
heapq.heapify(h)
heapq.heappush(h, 4)
a = heapq.heappop(h)
heapq.heappush(h, 1)
b = heapq.heappop(h)
c = heapq.heappop(h)
print(a, b, c, len(h))
```

a	b	c	len(h)

Zadanie 5. (0–2 pkt) ..... / 2

Program biblioteki szkolnej obsługuje 50 000 czytelników. Dla każdego wprowadzonego numeru karty sprawdza, czy taki numer istnieje. Uczeń trzyma numery w liście i sprawdza warunek numer in karty. Wskaż lepszą strukturę danych z biblioteki standardowej Pythona i uzasadnij wybór, porównując czas jednego sprawdzenia w obu strukturach.

Miejsce na odpowiedź

PIERWSZA ŁAWKA

Zadanie 5. (0–2 pkt) ..... / 2

Przeanalizuj program. Uzupełnij.

```
import heapq
h = []
for x in [2, 9, 6]:
    heapq.heappush(h, (x % 3, x))
print(heapq.heappop(h))
```

1. Program wypisze _____
2. Kopiec h po zakończeniu zawiera _____ elementy.

Zadanie 6. (0–3 pkt) ..... / 3

Aplikacja do wypożyczania laptopów musi wiele razy na sekundę sprawdzać, czy numer seryjny laptopa jest na liście zablokowanych (20 000 numerów). Którą strukturę danych wybierzesz – listę czy zbiór (set)? Uzasadnij, podając czas sprawdzenia w obu przypadkach.

Miejsce na odpowiedź

Zadanie 7. (0–4 pkt) ..... / 4

Na gałęzi main wykonano git merge sklepik i Git zgłosił konflikt w pliku ustawienia.py (fragment poniżej). Zespół ustalił, że obowiązuje wartość z gałęzi sklepik. Wyjaśnij, skąd pochodzi każdy z dwóch wariantów, zapisz, jak ma wyglądać ten fragment pliku po rozwiązaniu konfliktu, i podaj dwa polecenia, które kończą scalanie.

```
<<<<<<< HEAD
CENA = 4.50
=======
CENA = 5.00
>>>>>>> sklepik
```

Miejsce na odpowiedź

PIERWSZA ŁAWKA

CO JESZCZE JEST W ŚRODKU

Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – kartkówka, sprawdzian i karty pracy z powtórką,...

Podgląd pokazał 10 z 30 stron. Pełna wersja: 30 stron A4, PDF do druku i DOCX do edycji.

- Karta 1: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – najważniejsze wiadomości – 4 zadania, 8 pkt
- Karta 2: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – ćwiczenia podstawowe – 6 zadań, 19 pkt
- Karta 3: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – utrwalenie – 6 zadań, 11 pkt
- Karta 4: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – w praktyce, przy komputerze – 1 zadanie, 7 pkt
- Karta 5: Kartkówka: struktury danych i Git – grupa A – 4 zadania, 7 pkt
- Karta 6: Kartkówka: struktury danych i Git – grupa B – 4 zadania, 7 pkt
- Karta 7: Sprawdzian: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – grupa A – 8 zadań, 21 pkt
- Karta 8: Sprawdzian: Programowanie ZR: struktury danych i biblioteki, IDE, projekt zespołowy (repozytorium) – grupa B – 8 zadań, 21 pkt Karta 1 przypomina teorię i pokazuje przykła

Kup pełną wersję – plik od razu na e-mail

pierwszalawka.pl/material/informatyka-liceum-klasa-3-zr-dzial-11-programowanie-zr-struktury-danych-i-biblioteki-id