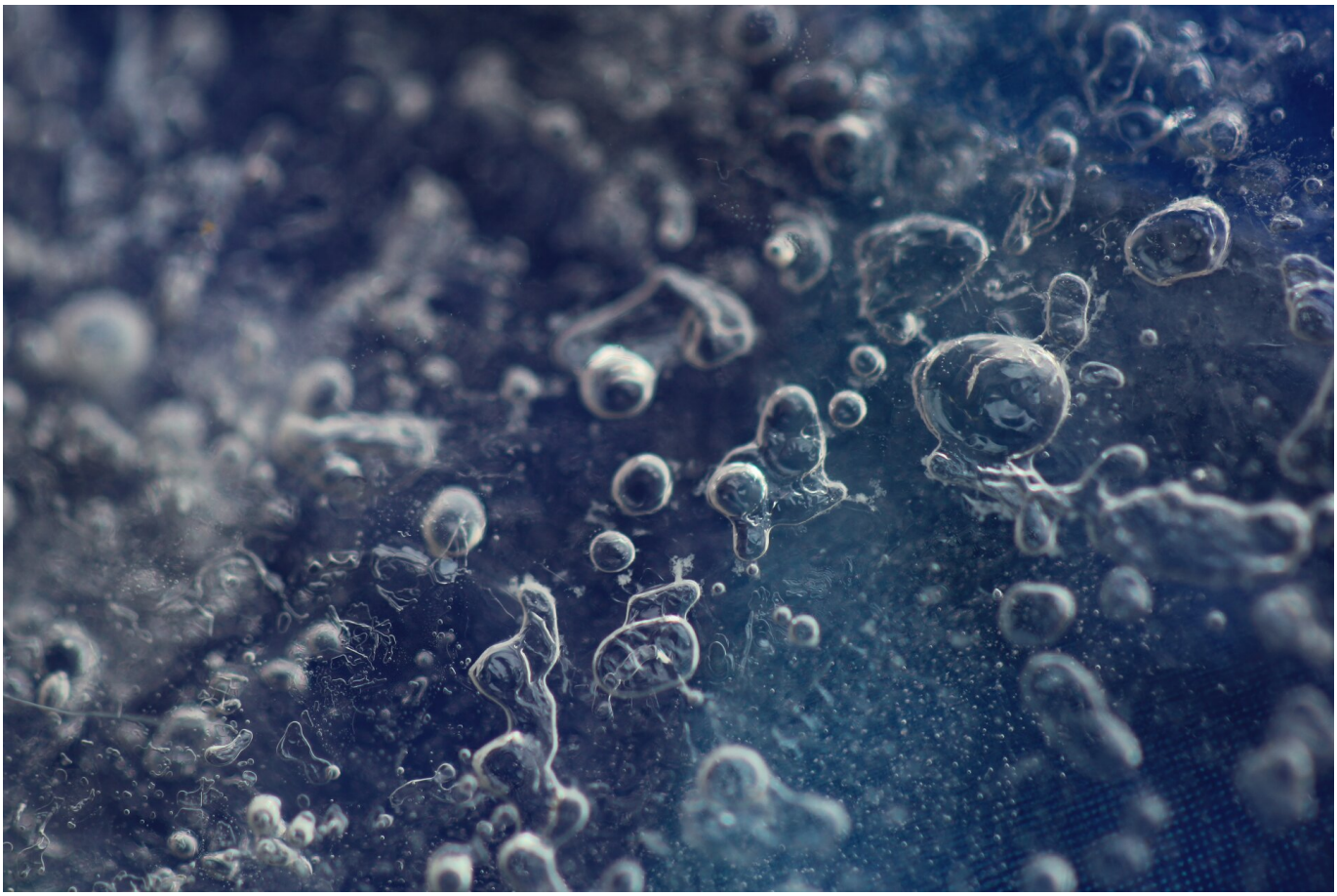
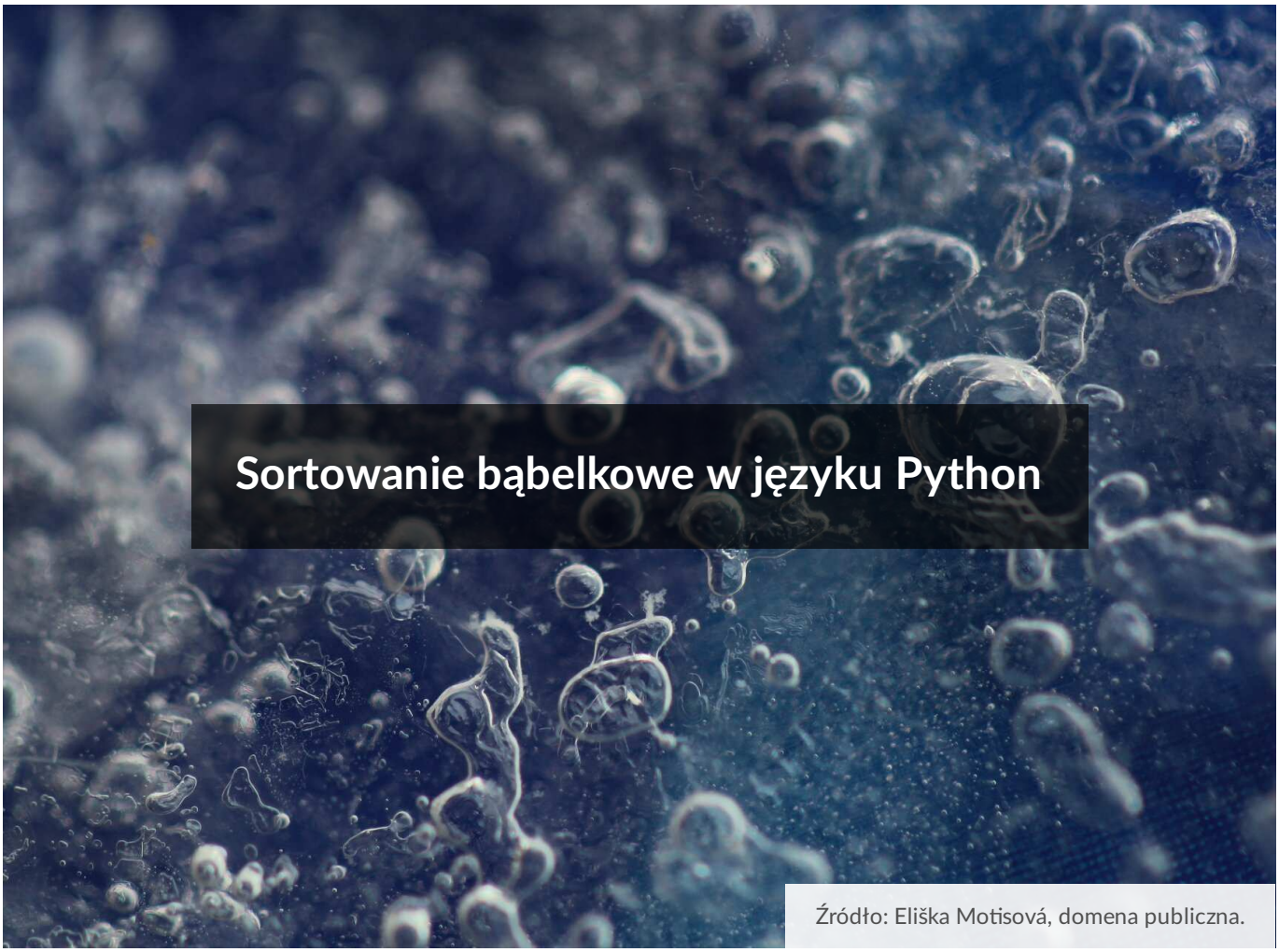


Sortowanie bąbelkowe w języku Python



Źródło: Eliška Motisová, domena publiczna.

Znasz już różne algorytmy sortowania. Jedne charakteryzują się mniejszą, inne większą złożonością czasową. Algorytm sortowania bąbelkowego należy do drugiej grupy — jego złożoność czasowa jest rzędu $O(n^2)$. Podstawowe informacje na jego temat znajdziesz w e-materiale Sortowanie bąbelkowe. Tutaj natomiast zajmiemy się jego implementacją w języku Python.

Implementacja algorytmu w innych językach programowania została omówiona w e-materiałach:

- Sortowanie bąbelkowe w języku C++,
- Sortowanie bąbelkowe w języku Java.

Więcej zadań? Zajrzyj do: Sortowanie bąbelkowe – zadania maturalne.

> **Twoje cele**
>
> - Przeanalizujesz działanie algorytmu oraz programu wykorzystującego sortowanie bąbelkowe.
> - Uporządkujesz różne typy danych z użyciem sortowania bąbelkowego.
> - Rozwiążesz przykładowe zadania z wykorzystaniem sortowania bąbelkowego.

Przeczytaj

Implementacja algorytmu sortowania bąbelkowego w języku Python

Polecenie 1

Zapoznaj się z prezentacją przedstawiającą krok po kroku implementację sortowania bąbelkowego w języku Python. Zastanów się, jak można zmodyfikować kod, aby zamiast sortować zbiór liczb, algorytm ten sortował łańcuchy znaków względem ich długości (od najkrótszego do najdłuższego lub odwrotnie).

1

Posortujemy niemalejąco n-elementową tablicę liczb całkowitych. Analogiczny algorytm może zostać użyty również do posortowania znaków, przy założeniu, że przyporządkowana jest im wartość liczbowa (np. kod w tablicy ASCII), na podstawie której będziemy je porównywać.

Program przetestujemy dla pięcioelementowego ciągu liczb całkowitych zapisanych w tablicy `tablica = [4, 2, 1, 5, 3]`.

Specyfikacja problemu:

Dane:

- `tablica` – n-elementowa tablica liczb całkowitych
- n – rozmiar tablicy; liczba naturalna

Wynik:

- `Tablica` z posortowanymi elementami w porządku niemalejącym

2 Implementację rozpoczynamy od deklaracji zmiennych z danymi wejściowymi.

3 Przechodzimy do zasadniczej części programu. Tworzymy pętlę zewnętrzną odpowiedzialną za operacje przechodzenia tablicy i porównywania jej elementów. Wykorzystujemy pętlę `for`. Wykona się ona $n - 1$ razy, co gwarantuje, że odpowiednie elementy sortowanej tablicy będą przetwarzane dopóty, dopóki wszystkie liczby w tablicy zostaną posortowane. W każdym cyklu największy element sortowanego ciągu zostanie umieszczony na końcu tablicy. Oznacza to, że największy element podciągu wymagającego sortowania w danej iteracji pętli znajdzie się w odpowiednim miejscu tablicy. Po powtórzeniu tego procesu $n - 1$ razy sortowanie zostanie zakończone, a wszystkie elementy zbioru będą się znajdowały w odpowiednich miejscach tablicy.

4 Tworzymy pętlę wewnętrzną, zagnieżdżoną w pętli zewnętrznej. Wartości iteratora pętli wewnętrznej będą wykorzystywane do wyznaczenia indeksów porównywanych elementów. Porównujemy liczbę zapisaną w tablicy pod wyznaczonym indeksem z elementem następnym, dlatego maksymalna wartość iteratora pętli wewnętrznej musi być o 1 mniejsza od liczby elementów sortowanego ciągu.

W każdym cyklu pętli zewnętrznej największy element zostaje umieszczony na właściwym

miejscu, dlatego od maksymalnej wartości iteratora pętli wewnętrznej odejmujemy wartość iteratora pętli zewnętrznej (w kodzie zmienna `i`), bo tyle elementów zostało już posortowanych. Wartość iteratora pętli wewnętrznej jest więc z dołu ograniczona przez `0` (przedział zamknięty – wskazuje pierwszy element tablicy), a z góry przez `n - i - 1` (przedział otwarty – tyle razy pętla ma się wykonać, ale żeby tak się stało, iterator nie może przyjmować tej wartości).

5

Aby posortować elementy ciągu przechowywanego w tablicy za pomocą algorytmu sortowania bąbelkowego, porównujemy pary sąsiadujących ze sobą elementów. Operację tę realizujemy w pętli wewnętrznej za pomocą instrukcji warunkowej `if`. Algorytm zgodnie z założeniami sortuje elementy niemalejąco, w związku z tym zamiana elementów jest przeprowadzona, gdy element wyznaczony przez iterator pętli wewnętrznej jest większy niż element występujący po nim w tablicy.

6

Zamianę elementów przeprowadzimy, używając zmiennej pomocniczej `pomoc`. Przypisujemy do niej wartość jednego z zamienianych elementów (w celu uniknięcia nadpisania tej wartości). Następnie do komórki tablicy, w której znajdowała się ta wartość, przypisujemy wartość drugiego elementu. W kolejnym kroku do komórki tablicy, w której do tej pory znajdowała się wartość drugiego elementu,

przypisujemy wartość przechowywaną w zmiennej pomocniczej.

7

Zapisany algorytm jest już w stanie poprawnie posortować tablicę. Zauważmy jednak, że w obecnej postaci programu zewnętrzna pętla zawsze wykona się dokładnie n – 1 razy – niezależnie od tego, czy tablica została już posortowana czy też nie. Spróbujmy usprawnić działanie algorytmu.

8

Aby usprawnić działanie programu, wykorzystamy zmienną logiczną `wykonanie`, w której przechowamy informację, czy w danej iteracji pętli zewnętrznej przeprowadzono operację zamiany elementów. Na początku każdej iteracji pętli zewnętrznej zmiennej przypisywana będzie wartość `False`. Natomiast wartość `True` przypiszemy zmiennej, gdy wykonana zostanie operacja przestawienia elementów. Zmienna `wykonanie` ma zmieniać wartość w przypadku zamiany elementów, co oznacza że kod ten musi znaleźć się wewnątrz instrukcji warunkowej `if`.

9

Zmienna `wykonanie` przyjmie wartość `True`, tylko jeżeli w jednej z iteracji pętli wewnętrznej nastąpiła operacja zamiany elementów. Oznacza to, że jeśli po zakończeniu działania pętli wewnętrznej zmienna ta ma wartość

`False`, to elementy tablicy są posortowane i możemy zakończyć działanie algorytmu. W tym celu tworzymy instrukcję warunkową `if`, w której będziemy sprawdzać wartość zmiennej `wykonanie`. W przypadku, gdy przyjmuje ona wartość `False`, przerwiemy działanie pętli zewnętrznej instrukcją `break`.

10

Tym samym udało się ukończyć program sortujący zapisany w tablicy ciąg liczb całkowitych niemalejąco za pomocą zoptymalizowanego algorytmu sortowania bąbelkowego. Cały kod prezentuje się następująco:

Dla zainteresowanych

Alternatywna implementacja algorytmu sortowania bąbelkowego w języku Python

Zapiszmy nieco inaczej program sortujący niemalejąco zadany ciąg liczb całkowitych. Tak ja w przypadku wcześniej zaprezentowanego algorytmu będziemy porównywać element wcześniejszy z następnym, zmienimy jedynie sposób indeksowania w pętli. W tym programie pętla będzie wyznaczała indeks elementu z prawej strony (dalszego) w danej parze. We wcześniejszej implementacji pętla wyznaczała indeks elementu z lewej strony (wcześniejszego) w danej parze.

1

Przedstawiona wcześniej implementacja nie jest jedyną poprawną wersją algorytmu sortowania bąbelkowego.

2

Inny wariant implementacji tego algorytmu zakłada porównywanie ze sobą elementu wyznaczonego przez wartość iteratora pętli wewnętrznej i liczby go poprzedzającej. Takie rozwiązanie wymaga modyfikacji w kodzie programu.

3

Pierwszą modyfikacją jest zmiana indeksów elementów tablicy, które będą ze sobą porównywane i ewentualnie zamieniane z `j` i `j + 1` na `j - 1` i `j`. Zmiana ta pociągnie za sobą również modyfikację wyrażenia logicznego w instrukcji warunkowej.

będzie wyglądał tak:

4

Drugą wymaganą modyfikacją jest zmiana ograniczeń wartości iteratora pętli wewnętrznej. Ponieważ porównujemy element z liczbą go poprzedzającą, to konieczne jest, by najmniejsza wartości przyjmowana przez iterator tej pętli wynosiła 1 (tak by wskazywał on na drugi element sortowanej tablicy). Z tego samego powodu górny limit wartości iteratora zwiększa się o 1 (skoro nie porównujemy liczby z elementem znajdującym po niej, to nie musimy zmniejszać górnego ograniczenia o 1, żeby nie wyjść z tablicy). Dozwolone wartości iteratora będą zatem ograniczone z góry przez wartość `n - i`, gdzie `i` to wartość iteratora pętli zewnętrznej. Fragment kodu z pierwszej implementacji:

będzie wyglądać następująco:

Cały kod drugiej wersji algorytmu prezentuje się następująco:

5

Słownik

algorytm

skończony ciąg zdefyniowanych czynności potrzebnych do wykonania zadań

inkrementacja

zwiększenie wartości zmiennej o 1

Gra edukacyjna

Polecenie 1

Sprawdź swoją wiedzę, odpowiadając na pytania zawarte w grze edukacyjnej.

Test

Sprawdź swoją wiedzę, biorąc udział w grze.

Poziom trudności:

łatwy

Limit czasu:

7 min

Twój ostatni wynik:

-

Uruchom

Polecenie 2

Zastanów się, które pytania sprawiły ci trudność. Wróć do fragmentów materiału, których dotyczą.

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Uzupełnij algorytm sortowania bąbelkowego, tak aby sortował on tablicę niemalejąco.

Przetestuj działanie programu dla następującej siedmioelementowej tablicy:

```
tablica = [5, 1, 4, 7, 694, 368, 874]
```

Specyfikacja problemu:

Dane:

- `tablica` – n-elementowa tablica liczb całkowitych
- n – liczba elementów w tablicy; liczba naturalna

Wynik:

- `tablica` – n-elementowa tablica liczb całkowitych posortowanych w kolejności niemalejącej

Twoje zadania

1. Uzupełnij algorytm sortowania bąbelkowego. Program powinien wyświetlać posortowaną niemalejąco tablicę `tablica`. Przetestuj program dla następującej tablicy: `[5, 1, 4, 7, 694, 368, 874]`.

Ćwiczenie 2

Napisz program sortujący daną tablicę nierosnąco, używając sortowania bąbelkowego.

Przetestuj działanie programu dla następującej tablicy:

```
tablica = [5, 1, 123, 4, 7, 432, 6]
```

Specyfikacja problemu:

Dane:

- `tablica` – n-elementowa tablica liczb całkowitych
- n – liczba elementów w tablicy; liczba naturalna

Wynik:

- `tablica` – n-elementowa tablica liczb całkowitych posortowanych nierosnąco

Twoje zadania

1. Napisz program sortujący daną tablicę nierosnąco, używając sortowania bąbelkowego. Przetestuj jego działanie dla następującej tablicy: `[5, 1, 123, 4, 7, 432, 6]`.

Ćwiczenie 3

W pewnej kręgielni zapisywano najlepsze dzienne wyniki. Po tygodniu przyszedł czas wyznaczenia rankingu graczy. Użyj sortowania bąbelkowego, aby wydrukować posortowaną niemalejąco listę wyników oraz graczy. Użyj wersji zoptymalizowanej algorytmu, w której w przypadku braku zmian w pojedynczym obrocie zewnętrznej pętli algorytm jest przerywany.

Specyfikacja problemu:

Dane:

- n – liczba graczy; liczba naturalna
- `wyniki` – wyniki poszczególnych graczy; n-elementowa tablica liczb całkowitych
- `gracze` – imiona graczy, których wyniki zarejestrowane są w systemie; i-temu graczowi tej tablicy odpowiada i-ty wynik z tablicy `wyniki`; n-elementowa tablica ciągów znaków

Wynik:

Wypisany ranking graczy i ich wyników w formie:

```
gracz_1: wynik_1
gracz_2: wynik_2
...
gracz_n: wynik_n
```

> **Ważne!**
>
> Pamiętaj, aby rezultaty były posortowane niemalejąco!

Twoje zadania

1. Program wyświetla posortowane niemalejąco wyniki wraz z graczami, którzy je zdobyli.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Sortowanie bąbelkowe w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy

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 podstawowy. Uczeń:

1) planuje kolejne kroki rozwiązywania problemu, z uwzględnieniem podstawowych etapów myślenia komputacyjnego (określenie problemu, definicja modeli i pojęć, znalezienie rozwiązania, zaprogramowanie i testowanie rozwiązania).

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

c) porządkowania ciągu liczb: przez wstawianie i metodą bąbelkową,

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;

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):

- Przeanalizujesz działanie algorytmu oraz programu wykorzystującego sortowanie bąbelkowe.
- Uporządkujesz różne typy danych z użyciem sortowania bąbelkowego.
- Rozwiążesz przykładowe zadania z wykorzystaniem sortowania bąbelkowego.

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. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Sortowanie bąbelkowe w języku Python". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu, np.
   – do czego służy algorytm sortujący?
   – co to jest dekrementacja?
   Chętni lub wybrani uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie w parach analizują implementacje algorytmów przedstawionych w sekcji „Przeczytaj". Następnie testują je na swoich komputerach.
2. **Praca z multimedium.** Uczniowie indywidualnie rozwiązują quiz z sekcji „Gra edukacyjna".
3. **Ćwiczenie umiejętności.** Uczniowie w parach wykonują ćwiczenia nr 1–2 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność wykonanych zadań, omawiając je wraz z uczniami.

Faza podsumowująca:

1. Nauczyciel wyświetla na tablicy temat lekcji i cele zawarte w sekcji „Wprowadzenie". W kontekście ich realizacji podsumowuje przebieg zajęć, a także wskazuje mocne i słabe strony pracy uczniów.
2. Wybrany uczeń podsumowuje zajęcia, zwracając uwagę na nabyte umiejętności, omawia ewentualne problemy podczas rozwiązania ćwiczeń.

Praca domowa:

1. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się".
2. Uczniowie wykonują polecenie 2 z sekcji „Gra edukacyjna".

Materiały pomocnicze:

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

Wskazówki metodyczne:

- Uczniowie mogą wykorzystać treści w sekcjach: „Przeczytaj”, „Gra edukacyjna”, „Sprawdź się” jako materiał do lekcji powtórkowej.