Anagramy w języku Python

Źródło: Wokandapix, domena publiczna.

Tworzenie anagramów jest zabawą i ćwiczeniem dla ludzkiego mózgu. Zadania takie często pojawiają się w czasopismach z szaradami (np. podaj anagram do słowa „poczytny” - rozwiązanie to słowo „optyczny”). Więcej na ich temat znajdziesz w e-materiale Anagramy.

W tym e-materiale zajmiemy się właśnie anagramami. Przy okazji nauczymy się wykorzystywać w języku Python funkcje służące do przetwarzania tekstu.

Implementację w pozostałych językach programowania znajdziesz w e-materiałach:

- Anagramy w języku C++,
- Anagramy w języku Java.

Więcej zadań? Przejdź do e-materiału Anagramy – zadania maturalne.

> **Twoje cele**
>
> - Poznasz funkcje wyrażeń regularnych z modułu `re` operujące na znakach.
> - Zastosujesz algorytm testujący, czy dwa słowa są anagramami.
> - Napiszesz funkcję generującą anagramy słów podanych jako argumenty.

Przeczytaj

Anagram jest słowem lub wyrażeniem, które powstaje w wyniku przestawienia liter lub sylab innego wyrazu lub innej frazy. Przykładami anagramów są:
`'krab' = 'brak' = 'bark'`.

Dla zainteresowanych

Więcej informacji o anagramach można znaleźć w serwisach poświęconych grze scrabble.

Przykład 1

W celu sprawdzenia czy różne ciągi znaków są anagramami możemy wykorzystać wbudowaną funkcję Pythona `sorted()`. Zwraca ona uporządkowaną rosnąco listę elementów obiektu (ciągu znaków), który został jej przekazany jako parametr.

Pamiętajmy o tym, jak działa funkcja `sorted()`:

- powtarzające się znaki są prezentowane jako kolejne elementy listy,
- kolejność znaków odzwierciedla ich pozycję w tablicy ASCII.
- narodowe znaki diakrytyczne są umieszczone za literami alfabetu łacińskiego.
- może posiadać parametr `reverse`, który pozwala zwrócić listę posortowaną w kolejności malejącej, np. `sorted("BED", reverse=True) -> "DEB"`,

Zapiszmy kod, który będzie sprawdzał, czy dwa ciągi znaków są anagramami:

```
1 def sprawdz_anagram_slow(slowo1, slowo2):
2     return sorted(slowo1) == sorted(slowo2)
3
4 print(sprawdz_anagram_slow('bark', 'krab'))
5 True
6
7 print(sprawdz_anagram_slow('bark', 'krat'))
8 False
```

Polecenie 1

Bazując na powyższym przykładzie i przykładach z lekcji o palindromach przygotuj taką modyfikację funkcji, aby wielkość znaków w sprawdzanych wyrażeniach nie miała znaczenia.

Przykład 2

Aby sprawdzić, czy zdania są anagramami, postępujemy podobne. Musimy jednak usunąć wszystkie znaki przestankowe (mogą one wpływać na wynik porównania ciągów).

Pamiętajmy o tym, jak działa funkcja `sorted()`:

- powtarzające się znaki są prezentowane jako kolejne elementy listy,
- kolejność znaków odzwierciedla ich pozycję w tablicy ASCII,
- narodowe znaki diakrytyczne są umieszczone za literami alfabetu łacińskiego.
- może posiadać parametr `reverse`, który pozwala zwrócić listę posortowaną w kolejności malejącej, np. `sorted("BED", reverse=True) -> "DEB"`,

Przygotujmy kod, który będzie sprawdzał, czy dwa zdania są anagramami:

```
def anagram_re(tekst1, tekst2):
    import re

    # zamianiamy znaki przestankowe na "" (nic)
    tekst1 = re.subn(r"[ ;,:.!?]", "", tekst1)
    # przypisujemy pierwszy element z tupli -> str
    tekst1 = tekst1[0]
    # zamieniamy na małe litery cały napis
    tekst1 = tekst1.lower()
    # wersja analogiczna jak powyżej, ale w jednym poleceniu
    tekst2 = re.subn(r"[ ;,:.!?]", "", tekst2)[0].lower()

    return sorted(tekst1) == sorted(tekst2)
```

Generowanie anagramów

Gdy chcemy utworzyć anagram dowolnego słowa, możemy posłużyć się prostym algorytmem:

1. Słowo zamieniamy w listę znaków, nie korzystając jednak z funkcji `sorted()` – powinniśmy bowiem zachować kolejność liter.
2. Z otrzymanej listy wybieramy losowo znak, który będzie kolejną (początkowo pierwszą) literą nowego słowa.
3. Z listy znaków usuwany wybraną w punkcie 2. literę.
4. Powtarzamy czynności opisane w punktach 2. i 3. dopóty, dopóki pozostają litery do wylosowania.

Przykład 3

Spróbujmy napisać program, który bazując na powyższym algorytmie utworzy anagram podanego słowa. Posłużymy się pętlą `while`.

```
def stworz_anagram(slowo):
    from random import randint
    lista = []

    for litera in slowo:
        lista.append(litera)

    anagram = ''

    while lista:
        losowana = randint(0, len(lista) - 1)
        nowa_litera = lista[losowana]
        del(lista[losowana])
        anagram += nowa_litera

    return anagram

# kilka przykładowych wywołań
print(stworz_anagram('bark'))
'karb'
print(stworz_anagram('bark'))
'akbr'
print(stworz_anagram('bark'))
'rkba'
print(stworz_anagram('bark'))
'arbk'
print(stworz_anagram('bark'))
'rbak'
print(stworz_anagram('bark'))
'brak'
```

Dla zainteresowanych

Oto inny wariant funkcji generującej anagramy – pokazuje on, jak w Pythonie można zminimalizować ilość kodu:

```
def stworz_anagram_2(slowo):
    from random import randint
```

```
    lista = [litera for litera in slowo]
    anagram = ''

    while lista:
        anagram += lista.pop(randint(0, len(lista) - 1))

    return anagram

# przykładowe wywołania
print(stworz_anagram('bark'))
'rkba'
print(stworz_anagram('bark'))
'arbk'
```

Już wiesz

Podsumujmy najważniejsze elementy tej lekcji:

- funkcja `sorted()` zwraca posortowano rosnąco listę elementów podanych jako pierwszy parametr,
- funkcja `sorted()` może posiadać parametr `reverse`, który pozwala zwrócić listę posortowaną w kolejności malejącej, np. `sorted("BED", reverse=True) -> "DEB"`,
- metoda `L.pop(x)` dla elementu `L` typu `list` pobiera element o indeksie `x` i kasuje go z listy.

Słownik

ASCII

(ang. American Standard Code for Information Interchange) pierwotnie siedmiobitowy system kodowania znaków (współcześnie rozszerzony do ośmiu bitów); w oryginalnej wersji kodom z zakresu 0–127 przyporządkowano 26 liter łacińskich, 10 cyfr (0...9) oraz dodatkowe znaki

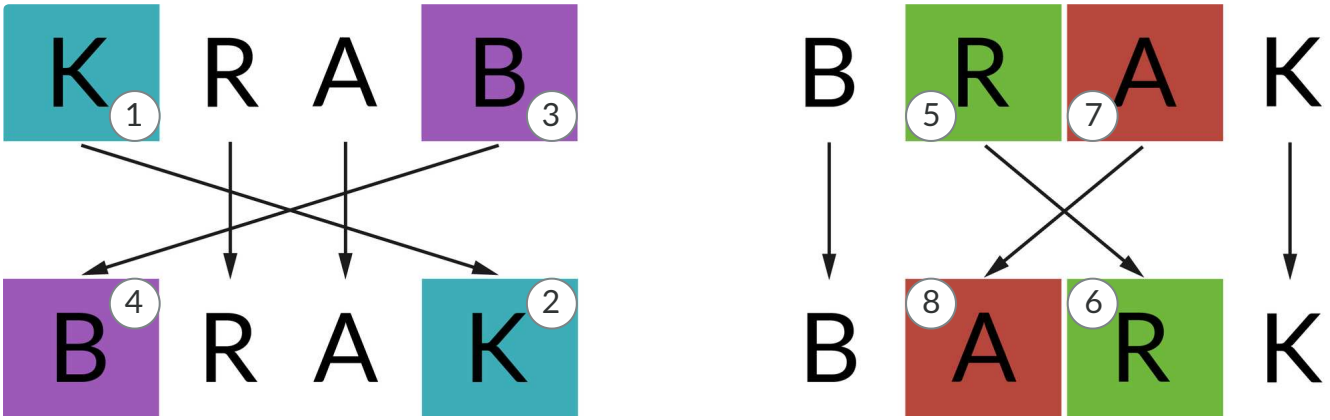
znaki diakrytyczne

wszystkie symbole (np. kropki, przecinki, dwukropki), które towarzyszą znakowi podstawowemu, zmieniając sposób odczytywania litery (w języku polskim są to np. kropki i kreski umieszczane nad literami 'ś' lub 'ż')

Schemat interaktywny

Polecenie 1

Przeanalizuj przykład tworzenia anagramu.



1

Litera "K"

z pozycji 1. zostanie przesunięta na pozycję 4.

2

Litera "K"

z pozycji 1. została przesunięta na pozycję 4.

3

Litera "B"

z pozycji 4. zostanie przesunięta na pozycję 1.

4

Litera "B"

z pozycji 4. została przesunięta na pozycję 1.

5

Litera "R"

z pozycji 2. zostanie przesunięta na pozycję 3.

6

Litera "R"

z pozycji 2. została przesunięta na pozycję 3.

7

Litera "A"

z pozycji 3. zostanie przesunięta na pozycję 2.

8

Litera "A"

z pozycji 3. została przesunięta na pozycję 2.

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

Polecenie 2

Przygotuj program, który sprawdzi czy dane słowa są swoimi anagramami, nie korzystaj z bloków umożliwiających sortowanie lub wbudowanych funkcji. Rozwiązanie zapisz w języku Python lub używając schematu blokowego.

Napisz program sprawdzający, czy dane słowa są swoimi anagramami.

```
# Tutaj dodaj własny kod. Użyj funkcji
# print()
```

```

```

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Uzupełnij podany kod, aby otrzymać funkcję sprawdzającą, czy podane dwa słowa są anagramem. Użyj funkcji `sorted`.

Specyfikacja:

Dane:

- `pierwsze_slowo, drugie_slowo` – zmienne typu *string*

Wynik:

Program na wyjściu standardowym zwróci wartość `True` lub `False`.

Twoje zadania

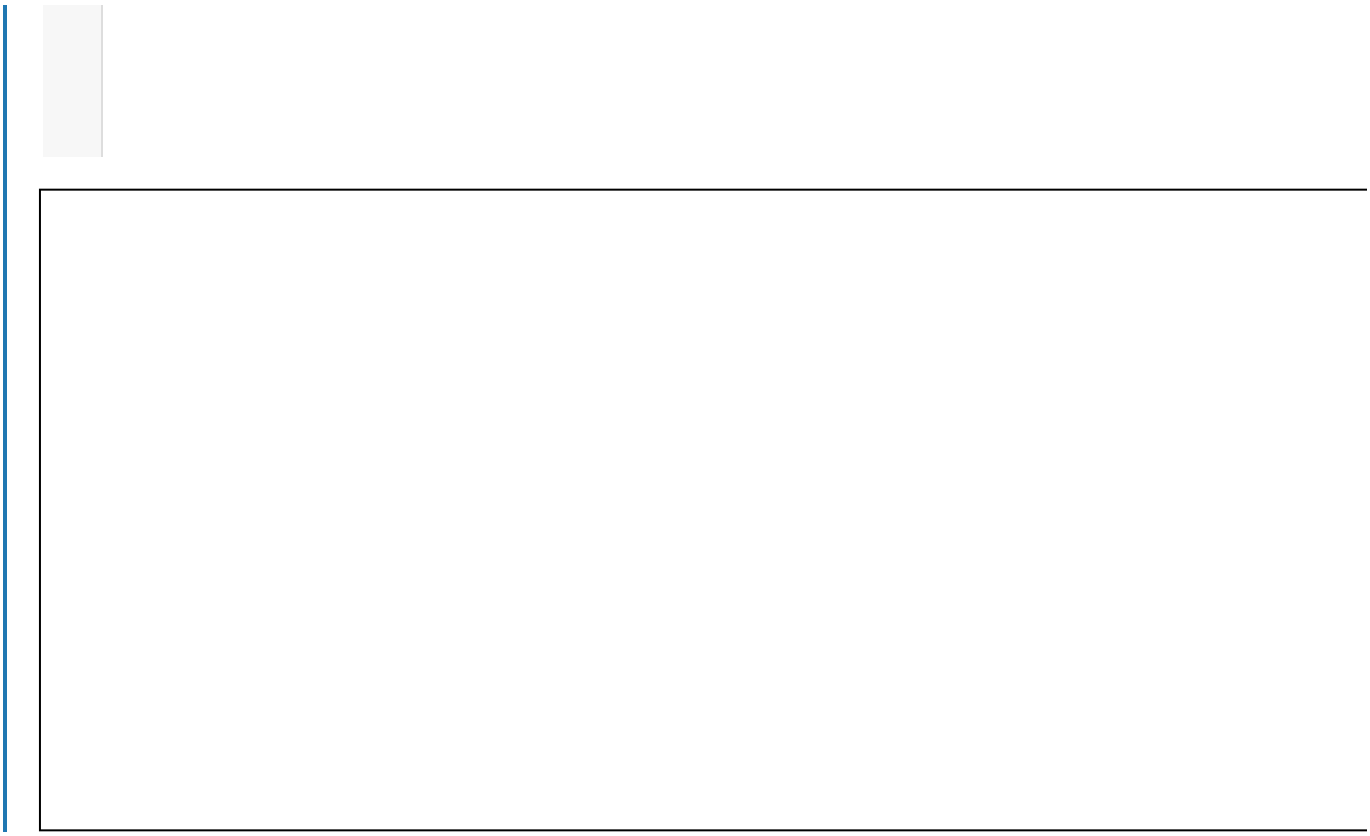
1. Zdefiniowanie funkcji `czy_anagram`
2. Sprawdzenie, czy funkcja dla argumentów `('krab', 'brak')` zwraca wartość `True`

```
def czy_anagram(pierwsze_slowo, drugie_slowo):
    # Tu uzupełnij kod
    pass

pierwsze_slowo = "krab"
drugie_slowo = "brak"

wynik = czy_anagram(pierwsze_slowo, drugie_slowo)
print(wynik)
```

```
1
```



Ćwiczenie 2

Uzupełnij podany kod, aby otrzymać funkcję sprawdzającą, czy podane dwa słowa są anagramem. Nie używaj funkcji `sorted`.

Specyfikacja:

Dane:

- `pierwsze_slowo, drugie_slowo` – zmienne typu *string*

Wynik:

Program na wyjściu standardowym zwróci wartośc `True` lub `False`.

Twoje zadania

1. Zdefiniowanie funkcji `czy_anagram`.
2. Funkcja zwraca `True` dla danych wejściowych `"alergia", "galeria"`.

```
def czy_anagram(pierwsze_slowo, drugie_slowo):
    litery_pierwsze_slowo = [0 for x in range(26)]
    litery_drugie_slowo = [0 for x in range(26)]

    # Tu uzupełnij kod

    for i in range(len(litery_pierwsze_slowo)):
        if litery_pierwsze_slowo[i] != litery_drugie_slowo[i]:
            return False
    return True

pierwsze_slowo = "alergia"
drugie_slowo = "galeria"
```

```
1
```

Ćwiczenie 3

Zdefiniujmy tablicę `slowa_bazowe`, która zawierać będzie słowa, które nie są swoimi anagramami. Napisz funkcję `czy_sa_anagramami`, która zwróci `True`, jeśli podane słowo jest anagramem któregokolwiek słowa z tablicy `slowa_bazowe`, oraz `False` w przeciwnym wypadku.

Specyfikacja:

Dane:

- `slowa_bazowe` – tablica łańcuchów znaków

Wynik:

Program na wyjściu standardowym zwróci wartośc `True` lub `False`.

Twoje zadania

1. Zdefiniowanie funkcji `czy_sa_anagramami`.
2. Funkcja zwraca `True` dla danych wejściowych `"tyran"`.

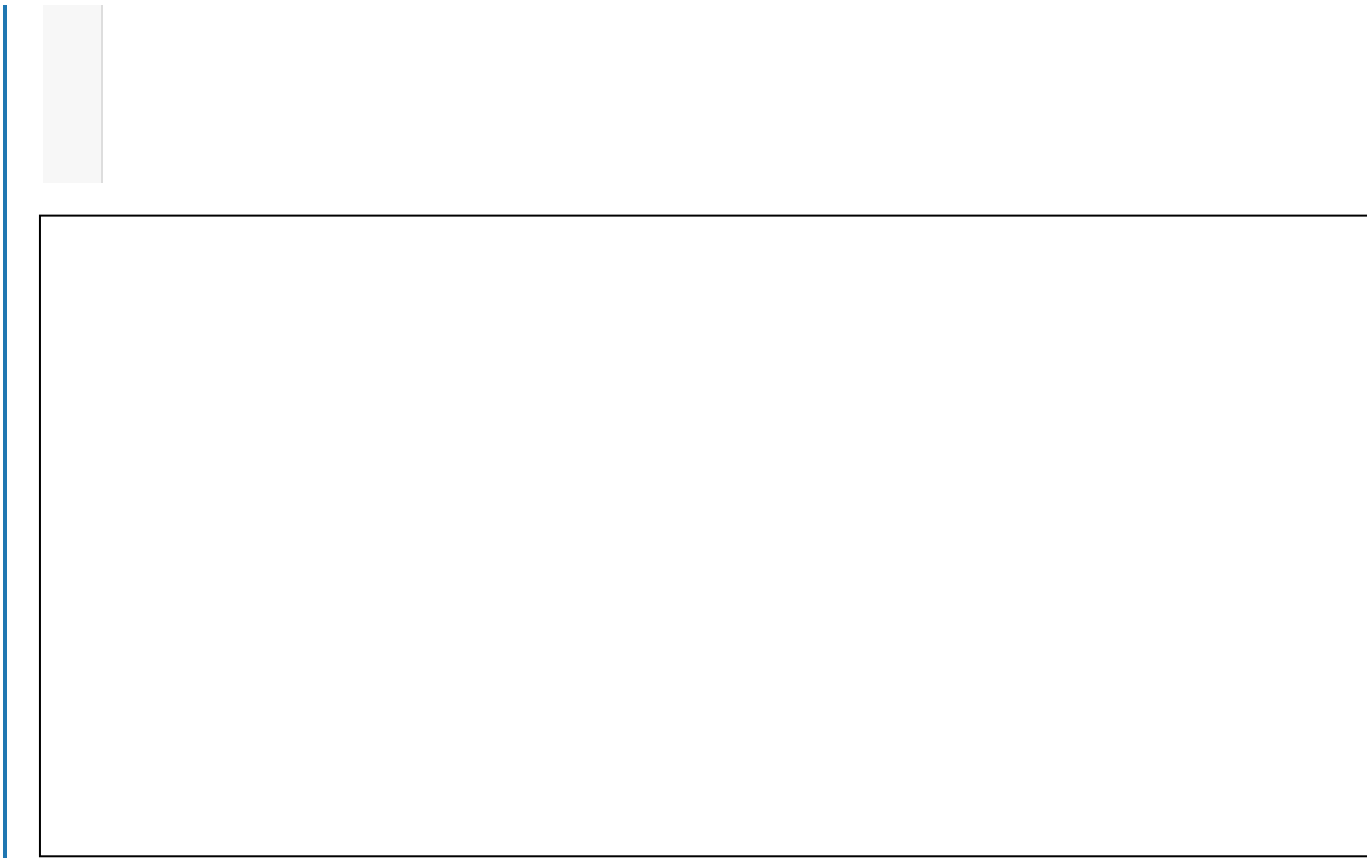
```
slowa_bazowe = ["mleko", "chleb", "narty", "krasa"]

def czy_sa_anagramami(slowo):
    # Tu uzupełnij kod
    pass

slowo = "tyran"

wynik = czy_sa_anagramami(slowo)
print(wynik)
```

```
1
```



Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Anagramy 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.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

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

b) na tekstach: porównywania tekstów, wyszukiwania wzorca w tekście metodą naiwną, szyfrowania tekstu metodą Cezara i przestawieniową,

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

Zakres podstawowy. Uczeń:

4) wyszukuje w sieci potrzebne informacje i zasoby, ocenia ich przydatność oraz wykorzystuje w rozwiązywanych problemach.

Cele operacyjne (językiem ucznia):

- uczeń zna i rozumie różnice między różnymi zastosowaniami pętli for
- uczeń potrafi tworzyć wyrażenia i instrukcje warunkowe
- uczeń tworzy algorytm i kod programu liczącego pensję od kwoty z różnymi procentami

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;
- telefony z dostępem do internetu;
- 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: „Anagramy w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematu i celów zajęć, przejście do wspólnego ustalenia kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Uczniowie tworzą pytania dotyczące tematu zajęć, na które odpowiedzą w trakcie lekcji.

Faza realizacyjna:

1. **Praca z tekstem.** Uczniowie analizują treści z sekcji „Przeczytaj" wyświetlone na tablicy.Następnie uczniowie w parach analizują przykłady zawarte w sekcji „Przeczytaj" oraz rozwiązują polecenie nr 1.

2. **Praca z multimedium.** Uczniowie w zespołach dwuosobowych zapoznają się z treścią polecenia nr 1 „W poniższym schemacie przygotuj algorytm, który sprawdzi czy dane słowa są swoimi anagramami, nie korzystaj z bloków umożliwiających sortowanie" z sekcji „Schemat interaktywny" i wspólnie analizują kolejne kroki rozwiązania postawionego problemu.
3. **Ćwiczenie umiejętności.** Uczniowie, pracując w parach, wykonują ćwiczenie nr 1 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność pisanych kodów, porównuje je i omawia wraz z uczniami. Wskazuje najbardziej efektywne rozwiązanie.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego uczniowie się nauczyli).
2. Wybrany uczeń podsumowuje zajęcia z programowania w Pythonie, zwracając uwagę na nabyte umiejętności.

Praca domowa:

1. Napisz program, który utworzy anagram dla danego słowa podanego przez użytkownika. Możesz wykorzystać analizę schematu tworzenia anagramu przedstawioną w Poleceniu 1 w sekcji „Schemat interaktywny".

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 multimedium w sekcji „Schemat interaktywny" do przygotowania się do lekcji powtórkowej.