Schemat Hornera w języku Python

- Wprowadzenie
- Przeczytaj
- Gra edukacyjna
- Sprawdź się
- Dla nauczyciela

Źródło: Charles Deluvio, domena publiczna.

W e-materiale Schemat Hornera poznaliśmy dwie wersje algorytmu, przy pomocy którego można obliczyć wartość wielomianu, wykorzystując minimalną liczbę mnożeń.

W tym e-materiale zaimplementujemy schemat Hornera w języku Python.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Schemat Hornera w języku C++,
- Schemat Hornera w języku Java.

Więcej zadań? Sięgnij do Schemat Hornera – zadania maturalne.

> **Twoje cele**
>
> - Zdefiniujesz funkcję obliczającą wartość wielomianu z wykorzystaniem metody iteracyjnej schematu Hornera.
> - Zdefiniujesz funkcję obliczającą wartość wielomianu z wykorzystaniem metody rekurencyjnej schematu Hornera.
> - Sprawdzisz, o ile mniej obliczeń trzeba wykonać, stosując schemat Hornera w celu wyznaczenia wartości niż stosując metodę tradycyjną.

Przeczytaj

Jak już wiemy, schemat Hornera jest wykorzystywany do obliczania wartości wielomianów. Wykorzystamy tę metodę w czasie bieżącej lekcji.

Za przykład posłuży nam wielomian trzeciego stopnia. Obliczymy jego wartość dla wybranych argumentów. Zrobimy to na dwa sposoby: najpierw metodą tradycyjną wykonamy wszystkie operacje, a następnie skorzystamy ze schematu Hornera. Ostatecznie porównamy ilości wykonywanych operacji.

Ogólna postać wielomianu trzeciego stopnia to:

$$W_{(x)} = a_0x^3 + a_1x^2 + a_2x + a_3$$

Możemy zapisać równanie obliczające jego wartość za pomocą wzoru:

$$W_{(x)} = (a_0 \cdot x \cdot x \cdot x) + (a_1 \cdot x \cdot x) + (a_2 \cdot x) + a_3$$

> **Ważne!**
>
> Wielomian, którego stopień jest równy n, składa się z $n + 1$ składników.

A oto przykładowy wielomian trzeciego stopnia:

$$W_{(X)} = 6x^3 + 8x^2 - 2x + 23$$

Dla `x = 2` jego wartość wynosi:

$$W_{(X=2)} = 6 \cdot 2^3 + 8 \cdot 2^2 - 2 \cdot 2 + 23 = 48 + 32 - 4 + 23 = 99$$

Aby obliczyć wynik, trzeba sześć razy wykonać operację mnożenia oraz trzykrotnie operacje dodawania:

$$W_{(X=2)} = 6 \cdot 2 \cdot 2 \cdot 2 + 8 \cdot 2 \cdot 2 - 2 \cdot 2 + 23 = 48 + 32 - 4 + 23 = 99$$

> **Przykład 1**
>
> Zapiszemy algorytm tradycyjny, dzięki któremu będziemy mogli obliczyć wartość takiego wielomianu. Ponieważ użyjemy pętli `for x in range(n+1)`, której wartość zmiennej

sterującej `x` zwiększa się od `0 do n`, więc współczynniki podamy jako listę elementów w kolejności od najmniejszego do największego wykładnika potęgi (od a_3 do a_0).

```
wielomian_tradycyjnie(stopien, wspolczynniki, argument)
    wynik przypisz 0
    aktualny_stopien przypisz 0

    wykonaj stopien+1 razy
        do wynik dodaj wspolczynniki[aktualny_stopien] * (argum
        zwiększ aktualny_stopien o 1

    wypisz wynik
```

Zdefiniujmy funkcję, która wykorzystując ten algorytm obliczy wartość wielomianu. Współczynniki towarzyszące poszczególnym jednomianom (czyli czynniki a_n w wyrażeniach $a_n x^n$) podamy w kolejności od najmniejszego do największego wykładnika potęgi (od a_3 do a_0):

```
def wielomian_iter(stopien, lista_wsp, argument):
    wynik = 0

    for _stopien in range(stopien + 1):
        wynik += lista_wsp[_stopien] * (argument ** _stopien)

    return f"Wynikiem dla argumentu {argument} jest wartość {wy

# przykład wykonania
print(wielomian_iter(3, [23, -2, 8, 6], 7))
Wynikiem dla argumentu 7 jest wartość 2459
```

Przykład 2

Zmodyfikujmy zapisaną funkcję w taki sposób, by obliczyć, ile operacji podstawowych było niezbędnych podczas wyznaczania wartości wielomianu. Sprawdźmy, czy uzyskany wynik zgadza się z wartością obliczoną na początku lekcji (powinien on wynosić 9).

```
def wielomian_iter_licz(stopien, lista_wsp, argument):
    wynik = lista_wsp[0]
    ile_oblicz = 0
```

```
4
5       for _stopien in range(1, stopien + 1):
6           wynik += lista_wsp[_stopien] * (argument ** _stopien)
7           ile_oblicz += _stopien # liczba mnożeń
8
9           ile_oblicz += 1 # liczba dodawań
10
11          return f"Aby obliczyć wielomian konieczne jest {ile_obl
12
13  # przykład wykonania
14  print(wielomian_iter_licz(3, [23, -2, 8, 6], 7))
15  Aby obliczyć wielomian konieczne jest 9 operacji podstawowych.
```

Obliczanie wartości wielomianu z zastosowaniem schematu Hornera

Schemat Hornera optymalizuje ilość operacji arytmetycznych niezbędnych do obliczenia wartości wielomianu.

Wykorzystując go otrzymujemy wzór:

$$W(_X) = (\ldots((a_0x + a_1)x + a_2)x + \ldots + a_{n-1})x + a_n$$

Kolejne współczynniki:

$$a_0 \ldots a_n$$

są mnożnikami potęg zmiennej x, zapisanymi od lewej do prawej strony wielomianu. Poszczególne wyrazy wielomianu (jednomiany) są zapisane od najwyższej potęgi zmiennej x do najniższej.

Po przekształceniu przykładowego wielomianu z wykorzystaniem schematu Hornera okazuje się, że w celu wyznaczenia wartości całego wyrażenia konieczne jest wykonanie trzech operacji mnożenia i trzech operacji dodawania:

$$W_{(X)} = (\ldots((a_0x + a_1)x + a_2)x + \ldots + a_{n-1})x + a_n$$

Przykład 3

Zapiszemy algorytm obliczania wartości wielomianu za pomocą wersji iteracyjnej schematu Hornera. W liście, która jest jednym z parametrów funkcji, współczynniki podajemy w kolejności od najmniejszego stopnia do największego, a przy niewystępujących potęgach w wielomianie uzupełniamy zerami.

```
wielomian_horner_iteracyjnie(stopien, wspolczynniki, argument)
    wynik przypisz wspolczynniki[stopien]

    wykonuj dopóki stopien > 0
        zmniejsz stopien o 1
        wynik przypisz wynik * argument + wspolczynniki[stopien

    zwróć wynik
```

Zdefiniujmy funkcję, która wykorzystując ten algorytm pozwoli nam obliczyć wartość wielomianu. Stopień wielomianu to najwyższa potęga `x' a` w tym wielomianie. W liście, która jest jednym z parametrów, współczynniki przy niewystępujących potęgach w wielomianie uzupełniamy zerami. Wówczas możemy pominąć podawanie stopnia jako parametru funkcji, gdyż będziemy mogli obliczyć go ze wzoru `stopien = liczba współczynników -1`.

```
def horner_iter(wsp, x):
    """Wersja iteracyjna algorytmu Hornera. Współczynniki w liś
    stopien = len(wsp) - 1
    wynik = wsp[stopien]

    while stopien > 0:
        stopien = stopien - 1
        wynik = wynik * x + wsp[stopien]

    return wynik

# przykładowe wykonanie
print(horner_iter([23, -2, 8, 6], 7))
2459
```

Możemy również sprawdzić, ile operacji podstawowych wykonujemy w tej funkcji:

```
def horner_iter_ilosc_oper(wsp, x):
```

```
    """Wersja iteracyjna algorytmu Hornera. Współczynniki w liś
    stopien = len(wsp) - 1
    wynik = wsp[stopien]
    zlicz = 0

    while stopien > 0:
        stopien = stopien - 1
        wynik = wynik * x + wsp[stopien]
        zlicz += 2 # 2 operacje podstawowe

    return wynik, zlicz

# przykładowe wykonanie
print(horner_iter([23, -2, 8, 6], 7))
(2459, 6)
```

Definiując funkcję rekurencyjną, będziemy musieli podać następujące dane wejściowe:

- stopień wielomianu,
- współczynniki kolejnych wyrazów wielomianu,
- argument, dla którego obliczamy wartość wielomian.

Przykład 4

Wzór rekurencyjny obliczania wartości wielomianu za pomocą schematu Hornera możemy zapisać w następujący sposób:

$$W_n = \begin{cases} a_0 & \text{dla } n = 0 \\ W_{n-1}(x) \cdot x + a_n & \text{dla } n \geq 1 \end{cases}$$

Zapiszemy algorytm rekurencyjny, który pozwala obliczyć wartość wielomianu z zastosowaniem schematu Hornera:

```
rek_horner(stopien, lista_wspolczynnikow, wartosc_argumentu)
    jesli stopien jest równy 0, zwróć lista_wspolczynnikow[0]
    w przeciwnym przypadku
        zwróć (wartosc_argumentu
        * rek_horner(stopien - 1, lista_wspolczynnikow, wartosc
        + lista_wspolczynnikow[stopien])
```

Zdefiniujmy funkcję, która wykorzystując algorytm rekurencyjny obliczy wartość wielomianu. Współczynniki towarzyszące poszczególnym jednomianom (czyli czynniki a_n w wyrażeniach $a_n x^n$) podamy w kolejności od największego do najmniejszego wykładnika potęgi (od a_0 do a_3):

```
def rek_horner(stopien, lista_wsp, argument):
    if stopien == 0:
        return lista_wsp[stopien]
    else:
        return argument * rek_horner(stopien - 1, lista_wsp, ar

# przykład wywołania
print(rek_horner(3, [6, 8, -2, 23], 7))
2459
```

Sprawdźmy, ile operacji podstawowych trzeba wykonać w celu obliczenia wartości wielomianu z zastosowaniem metody rekurencyjnej.

Ważne!

Funkcja rekurencyjna wywołuje wielokrotnie siebie samą; w celu sprawdzenia liczby wykonywanych przez nią operacji musimy posłużyć się zmienną, która istnieje w przestrzeni nazw poza funkcją. Do tego celu użyjemy zmiennej typu integer, który jest typem niezmiennym. Aby zmieniać wartość takiej zmiennej z wnętrza funkcji, musimy zadeklarować dostęp do niej za pomocą słowa kluczowego `global`. Dzięki temu będziemy mogli zmieniać wartość zmiennej, która istnieje „poza" funkcją, a jest niezmienna.

Przykład 5

Zdefiniujemy funkcję, która poda liczbę wykonanych operacji arytmetycznych podczas wyznaczania wartości wielomianu z zastosowaniem metody rekurencyjnej:

```
def rek_horner_licz(stopien, lista_wsp, argument):
    global zlicz

    if stopien == 0:
        return lista_wsp[stopien]
    else:
        zlicz += 2 # jedno wywołanie funkcji to 2 operacje pods
        return argument * rek_horner_licz(stopien - 1, lista_ws
```

Spróbujmy uruchomić funkcję; pamiętajmy o wcześniejszym zadeklarowaniu zmiennej `zlicz` oraz o wyświetleniu jej wartości po zakończeniu działania funkcji:

```
# inicjujemy zmienną, w której będziemy zliczać ilości wykonani
zlicz = 0

# uruchamiamy właściwą funkcję
rek_horner_licz(3, [6, 8, -2, 23], 7)

# sprawdzamy wartość zmiennej
print(zlicz)
6
```

Polecenie 1

Bazując na postaci rekurencyjnej funkcji obliczającej wartość wielomianu, spróbujmy zapisać ją używając wyrażenia trójargumentowego.

Dla zainteresowanych

Dla języka Python opracowano moduł o nazwie SymPy – pozwala on na zapisywanie wyrażeń algebraicznych i obliczanie ich wartości.

Słownik

William George Horner

brytyjski matematyk żyjący w latach 1786-1837; w wieku 14 lat został asystentem nauczyciela w szkole w Bristolu, a w 1809 roku założył własną szkołę w Bath; podał sposób wyznaczania wartości wielomianu z wykorzystaniem minimalnej liczby operacji możenia
wielomian

wyrażenie algebraiczne będące sumą jednomianów, czyli wyrażeń postaci <math$a_n x^n$
stopień wielomianu

najwyższa potęga przy zmiennej, która nie jest równa zero
algorytm

przepis postępowania prowadzący do rozwiązania ustalonego problemu, określający ciąg czynności elementarnych, które należy w tym celu wykonać
iteracja

technika programowania polegająca na powtarzaniu tych samych operacji w pętli określoną liczbę razy lub do momentu, aż zostanie spełniony zadany warunek
rekurencja

technika programowania polegająca na odwoływaniu się procedur lub funkcji do samych siebie, aż do momentu spełnienia warunku podstawowego
przypadek podstawowy

przypadek, dla którego funkcja rekurencyjna nie wywołuje samej siebie, a zamiast tego zwraca z góry określoną wartość
wykonanie elementarne w rekurencji

moment, gdy funkcja rekurencyjna zwraca konkretną wartość zamiast kolejnego wywołania rekurencyjnego
przestrzeń nazw

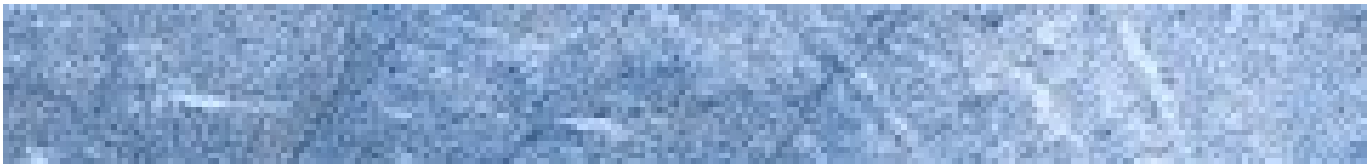
(ang. *namespace*) – miejsce w pamięci operacyjnej, w której przechowywana jest dana zmienna; najczęściej przestrzeń nazw związana jest z funkcją, a zmienne używane w niej są widoczne tylko w obrębie tej przestrzeni – jest to „zakres obowiązywania" tych zmiennych
niezmienna

(ang. *immutable*) sekwencja lub zmienna, która jest niemodyfikowalna „w miejscu", a więc nie można zmienić żadnego z jej elementów wewnątrz, trzeba stworzyć nowy obiekt, który zawiera zmienione elementy

Gra edukacyjna

Polecenie 1

Sprawdź swoją wiedzę biorąc udział w quizie.



Test

Sprawdź swoją wiedzę

Poziom trudności:

łatwy

Limit czasu:

4 min

Twój ostatni wynik:

-

Uruchom

Polecenie 2

Wstaw właściwe słowa w luki w tekście.

Inaczej plan to [ ].

Tupla po polsku to [ ].

W informatyce sposób postępowania/opis sposobu rozwiązania problemu to [ ].

Niniejsza lekcja mówi o [ ].

Polecenia to inaczej [ ].

Instrukcja yield wykorzystywana jest w [ ].

Algorytm | Wielomianach | Instrukcje | Schemat | Generatorze | Krotka

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Przedstawione zadanie zostało opracowane przez Centralną Komisję Egzaminacyjną i weszło w skład zbioru zadań przygotowujących do egzaminu maturalnego w roku 2015.

Wielomianem parzystym nazywamy wielomian stopnia 2n postaci:

$$R_{(x)} = a_n x^{2n} + a_{n-1} x^{2n-2} + \ldots + a_2 x^2 + a_0$$

tzn. taki, w którym występują tylko parzyste potęgi zmiennej.

Bazując na schemacie Hornera, zdefiniuj funkcję `testowa(stopien, lista_wsp, argument)`, która dla podanego zestawu parametrów oblicza wartość parzystego wielomianu.

Dane wejściowe:

- n - liczba całkowita `>= 0`
- x - liczba rzeczywista
- a . . . - liczby rzeczywiste

Wynik:

$$\text{Wartość } \mathrm{R}_{(x)}$$

Przykładowe wykonania:

```
print(testowa(3, [3, 2, 1, 0], 1))
6
print(testowa(3, [1, 1, 1, 0], 2.1))
109.624221
print(testowa(3, [1, 1, 1, 0], 5))
```

```
16275
print(testowa(3, [6, 8, -2, 23], 1))
35
print(testowa(3, [3, 2, 1, 0], 2))
228
```

Twoje zadania

1. Zdefiniowanie funkcji `testowa(stopien, lista_wsp, argument)`, działającej zgodnie z opisem.
2. Sprawdzenie, czy funkcja dla parametrów `3, [6, 8, -2, 23], 2` zwraca wartość 527.
3. Sprawdzenie, czy funkcja dla parametrów `3, [6, 8, -2, 23], 5` zwraca wartość 98723.

```
def testowa(stopien, lista_wsp, argument):
    # tutaj dodaj własny kod
    return None

# niżej podaj przykład wywołania funkcji
wynik = testowa(3, [6, 8, -2, 23], 2)
print(wynik)
print(wynik==527)
#
wynik = testowa(3, [6, 8, -2, 23], 5)
print(wynik)
print(wynik==98723)
```

```
```

Ćwiczenie 2

Zdefiniuj funkcję `suma_poteg(lista_elementow)`, która obliczy sumę potęg kolejnych elementów listy, gdzie podstawą będzie element, a wykładnikiem indeks tego elemenu w liście.

Elementy listy muszą być typu `int` lub `float`. Jeśli którykolwiek z elementów listy będzie innego typu, funkcja powinna zwrócić wartość `False`.

Przykładowe wywołania:

```
print(suma_poteg([2.01, 4.3, 5.4]))
34.46
print(suma_poteg([1, 22, 33, 5.01]))
1237.751501
print(suma_poteg([53245324.03, 2, 4.5, 5]))
148.25
print(suma_poteg([53245324.03, "A", 4.5, 5]))
False
```

Twoje zadania

1. Zdefiniowanie funkcji `suma_poteg()`.
2. Sprawdzenie, czy funkcja `suma_poteg()` poprawnie zwraca wartość `False` dla ustalonych parametrów.
3. Sprawdzenie, czy funkcja `suma_poteg()` poprawnie zwraca wartość dla ustalonych parametrów.

```
def suma_poteg(lista):
    # tutaj Twój kod
    return None

# tu przykładowe wywołania
print(suma_poteg([-2, -3, -4, 1]))
print(suma_poteg([-2, "3", -4, 1]))
print(suma_poteg([-2, -3.5, -4, 1.4]))
```

```
1
```

Przedstaw swoje uzasadnienie do kodu napisanej funkcji.

Przedstaw swoje uzasadnienie do kodu napisanej funkcji.

Ćwiczenie 3

Zdefiniuj funkcję `iloczyn_poteg_odwrotny(lista_elementow)`, która obliczy iloczyn potęg kolejnych elementów listy (w której przede wszystkim należy odwrócić ich kolejność), gdzie podstawą będzie element, a wykładnikiem indeks tego elementu w liście.

Elementy listy muszą być typu `int` lub `float`. Jeśli którykolwiek z elementów listy będzie innego typu, funkcja powinna zwrócić wartość `False`.

Przykładowe wywołania:

```
print(iloczyn_poteg_odwrotny([2, 0, 1]))
0
print(iloczyn_poteg_odwrotny([-2, -3, -4, "A"]))
False
print(iloczyn_poteg_odwrotny([2, 1, 1]))
4
print(iloczyn_poteg_odwrotny([2, 20.3, 324432]))
81.2
print(iloczyn_poteg_odwrotny([3, 4, 20, 0]))
8640
```

Twoje zadania

1. Zdefiniowanie funkcji `iloczyn_poteg_odwrotny()`.
2. Sprawdzenie, czy funkcja `iloczyn_poteg_odwrotny()` poprawnie zwraca wartość `False` dla ustalonych parametrów.
3. Sprawdzenie, czy funkcja `iloczyn_poteg_odwrotny()` poprawnie zwraca wartość dla ustalonych parametrów.

```
def iloczyn_poteg_odwrotny(lista):
    # tutaj Twój kod
    return None

# tu przykładowe wywołania
print(iloczyn_poteg_odwrotny([-2, -3, -4, 1]))
print(iloczyn_poteg_odwrotny([-2, "3", -4, 1]))
print(iloczyn_poteg_odwrotny([-2, -3.5, -4, 1.4]))
```

```
1
```

Przedstaw swoje uzasadnienie do kodu napisanej funkcji.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Schemat Hornera w języku Python

Grupa docelowa:

Liceum ogólnokształcące i technikum, liceum ogólnokształcące, technikum, zakres 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

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);

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

1) zapisuje za pomocą listy kroków, schematu blokowego lub pseudokodu, i implementuje w wybranym języku programowania, algorytmy poznane na wcześniejszych etapach oraz algorytmy:

h) obliczania wartości wielomianu za pomocą schematu Hornera,

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):

- Zdefiniujesz funkcję obliczającą wartość wielomianu z wykorzystaniem metody iteracyjnej schematu Hornera.
- Zdefiniujesz funkcję obliczającą wartość wielomianu z wykorzystaniem metody rekurencyjnej schematu Hornera.
- Sprawdzisz, o ile mniej obliczeń trzeba wykonać, stosując schemat Hornera w celu wyznaczenia wartości niż stosując metodę tradycyjną.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych.

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: „Schemat Hornera w języku Python". Nauczyciel prosi uczniów o zapoznanie się

z treściami w sekcji „Przeczytaj”.

Faza wstępna:

1. Wyświetlenie przez nauczyciela tematu i celów lekcji. Określenie wiążących dla uczniów kryteriów sukcesu.
2. **Rozpoznanie wiedzy uczniów.** Nauczyciel zadaje uczniom pytania dotyczące ich aktualnego stanu wiedzy w obszarze poruszanego tematu i programowania, np.
   – co to jest wielomian?
   – czym jest przestrzeń nazw?
   Chętni uczniowie udzielają na nie odpowiedzi.

Faza realizacyjna:

1. Uczniowie analizują przykład z sekcji „Przeczytaj” i powtarzają zaprezentowane rozwiązanie na swoim komputerze.
2. **Ćwiczenie umiejętności.** Uczniowie wykonują indywidualnie ćwiczenia nr 1-3 z sekcji „Sprawdź się”. Po wykonaniu każdego z nich następuje omówienie rozwiązania przez nauczyciela.

Faza podsumowująca:

1. Na koniec zajęć nauczyciel prosi uczniów o rozwinięcie zdania: „Na dzisiejszych zajęciach nauczyłam/łem się jak…”.

Praca domowa:

1. Uczniowie wykonują polecenie 1 oraz test 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.