Odwrotna notacja polska w języku Python

Bibliografia:

- Źródło: Kazimierz Trzęsicki, *Wkład logików polskich w światową informatykę*, „Filozofia Nauki” 2006, nr 14/3, s. 8–9.

Źródło: Guillaume Techer, domena publiczna.

Część języków programowania (a także kalkulatorów) używa do swoich obliczeń odwrotnej notacji polskiej. Wiesz już, na czym ona polega i czym różni się od tradycyjnego zapisu wyrażeń arytmetycznych.

W tym e-materiale zaimplementujesz algorytm konwersji z notacji infiksowej do ONP w języku Python.

Jeśli ciekawi cię, jak wyglądają implementacje w innych językach programowania, możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Odwrotna notacja polska w języku C++,
- Odwrotna notacja polska w języku Java.

Więcej zadań? Sięgnij do: Odwrotna notacja polska – zadania maturalne.

> **Twoje cele**
>
> - Wykorzystasz w praktyce sposób zapisywania wyrażeń arytmetycznych z zastosowaniem odwrotnej notacji polskiej.
> - Zaimplementujesz w języku Python algorytm przekształcania wyrażenia arytmetycznego zapisanego z wykorzystaniem notacji infiksowej do postaci w ONP.
> - Rozwiążesz przykładowe zadania wymagające wykazania się znajomością ONP.

Przeczytaj

Polecenie 1

Przeanalizuj prezentację przedstawiającą kolejne kroki implementacji rekurencyjnego algorytmu konwertującego wyrażenie arytmetyczne zapisane za pomocą notacji infiksowej na ONP. Następnie zastanów się, dlaczego program działa jedynie w sytuacji, gdy wyrażenie arytmetyczne zawiera liczby naturalne. Jak wyglądałby algorytm, który umożliwiałby wprowadzanie liczb całkowitych bądź rzeczywistych?

Specyfikacja problemu:

Dane:

- `klasyczna` – łańcuch znaków przechowujący wyrażenie arytmetyczne zapisane w notacji infiksowej, pozbawione spacji, gdzie każde z działań wydzielone jest parą nawiasów

Wynik:

- wyrażenie arytmetyczne zamienione na ONP

1

Zaimplementujemy w języku Python algorytm zamiany wyrażenia arytmetycznego zapisanego w postaci infiksowej na ONP. Pamiętaj, że sprawdza się on jedynie w przypadku wyrażeń arytmetycznych zawierających liczby naturalne.

2

Zaczniemy od deklaracji funkcji, która będzie zmieniać zapis wyrażenia arytmetycznego. Przyjmie ona dwa parametry: ciąg znaków zawierający analizowane wyrażenie

arytmetyczne oraz indeks `i`, odpowiadający pozycji aktualnie badanego elementu w podanym ciągu znaków.

```
def ONP(klasyczna, i):
    pass
```

3

Następnie zapiszemy w ciele funkcji instrukcję warunkową `if`. Będzie ona odpowiadać za wykonanie tej części kodu, która powinna zostać wywołana w sytuacji, gdy badany znak (ten o indeksie `i`) będzie różny od nawiasu otwierającego „(”. Jest to przypadek bazowy. Po natrafieniu na niego przerywamy rekurencję.

```
def ONP(klasyczna, i):
    if klasyczna[i] !=
'(':
        pass
    else:
        pass
```

4

Jeżeli znak o indeksie `i` nie jest znakiem „(”, oznacza to, że jest on liczbą i należy ją wypisać, a następnie zwrócić wartość indeksu. Ponieważ liczba może składać się z więcej niż jednej cyfry, do wypisania każdej z nich użyjemy pętli `while`.

```
def ONP(klasyczna, i):
    if klasyczna[i] !=
'(':
        while 0 <=
(ord(klasyczna[i]) -
ord("0")) <= 9:
            pass
    else:
```

```
6           pass
```

5

Dopóki kolejny znak jest cyfrą, wypisujemy go w konsoli i zwiększamy wartość zmiennej `i` o 1.

```
def ONP(klasyczna, i):
    if klasyczna[i] !=
'(':

        while 0 <=
(ord(klasyczna[i]) -
ord("0")) <= 9:

print(klasyczna[i], end =
"")
            i += 1

    else:
        pass
```

6

Po zakończeniu działania zmienna `i` będzie wskazywać na znak za wypisaną liczbą. Dlatego odejmujemy od niej 1, aby wskazywała ostatni element operandu.

```
def ONP(klasyczna, i):
    if klasyczna[i] !=
'(':

        while 0 <=
(ord(klasyczna[i]) -
ord("0")) <= 9:

print(klasyczna[i], end =
"")
            i += 1
```

```
7
8           print(" ", end =
    "")
9           return i - 1
10
11      else:
12          pass
```

7

Przejdźmy do uzupełnienia kodu, który powinien się wykonać, gdy wskazywany przez zmienną `i` znak w napisie `klasyczna` jest nawiasem otwierającym. Wywołujemy funkcję ONP z indeksem powiększonym o 1. Funkcja rekurencyjna będzie wykonywana tak długo, aż lewy operand tej operacji zostanie przeanalizowany.

```
1  def ONP(klasyczna, i):
2      if klasyczna[i] !=
    '(':
3
4          while 0 <=
    (ord(klasyczna[i]) -
    ord("0")) <= 9:
5
    print(klasyczna[i], end =
    "")
6              i += 1
7
8          print(" ", end =
    "")
9          return i - 1
10
11     else:
12         i = ONP(klasyczna,
    i + 1)
13
```

8

Wiedząc, że indeks `i` wskazuje pozycję ostatniego elementu pierwszego operandu wyrażenia arytmetycznego, możemy wywnioskować, że kolejny znak ciągu to operator arytmetyczny, który w odwrotnej notacji polskiej zapisywany jest na końcu wyrażenia arytmetycznego. Zamierzamy zapisać jego wartość do zmiennej `operator`. Zwiększymy więc wartość indeksu `i` o 1, a następnie pobierzemy znak operatora do zmiennej.

```
1  def ONP(klasyczna, i):
2      if klasyczna[i] !=
   '(':
3
4          while 0 <=
   (ord(klasyczna[i]) -
   ord("0")) <= 9:
5
   print(klasyczna[i], end =
   "")
6              i += 1
7
8          print(" ", end =
   "")
9          return i - 1
10
11     else:
12
13         i = ONP(klasyczna,
   i + 1)
14         i += 1
15         operator =
   klasyczna[i]
16
```

Konwersja wyrażenia arytmetycznego obliczana jest dla dwóch operandów umieszczonych

9

między nawiasami. W kodzie zawarliśmy już analizę pierwszego z nich – lewego. Teraz czas zająć się drugim, czyli prawym.

```
def ONP(klasyczna, i):
    if klasyczna[i] != '(':

        while 0 <= (ord(klasyczna[i]) - ord("0")) <= 9:

print(klasyczna[i], end = "")
            i += 1

        print(" ", end = "")
        return i - 1

    else:

        i = ONP(klasyczna, i + 1)
        i += 1
        operator = klasyczna[i]
        i = ONP(klasyczna, i + 1)

```

10

Teraz, gdy przeanalizowaliśmy już oba operandy, zwiększymy indeks `i` o 1, ponieważ następny znak będzie prawym nawiasem „)”, który nie wnosi nic do rozwiązania. Możemy go pominąć. Dodatkowo, ponieważ przeanalizowaliśmy już oba operandy, możemy wypisać na standardowe wyjście wartość zmiennej `operator`, a następnie zwrócić wartość zmiennej `i`. Posłuży ona kolejnemu wywołaniu funkcji do

kontynuowania analizy dalszych części wyrażenia arytmetycznego.

```
def ONP(klasyczna, i):
    if klasyczna[i] != '(':

        while 0 <= (ord(klasyczna[i]) - ord("0")) <= 9:

print(klasyczna[i], end = "")
            i += 1

        print(" ", end = "")
        return i - 1

    else:

        i = ONP(klasyczna, i + 1)
        i += 1
        operator = klasyczna[i]
        i = ONP(klasyczna, i + 1)
        i += 1

        print(operator, end = " ")
        return i

```

Dodajmy do programu przykładowe wywołanie.

11

```
def ONP(klasyczna, i):
    if klasyczna[i] != '(':

```

```
        while 0 <= (ord(klasyczna[i]) - ord("0")) <= 9:
            print(klasyczna[i], end = "")
            i += 1

        print(" ", end = "")
        return i - 1

    else:

        i = ONP(klasyczna, i + 1)
        i += 1
        operator = klasyczna[i]
        i = ONP(klasyczna, i + 1)
        i += 1

        print(operator, end = " ")
        return i

klasyczna = "((8*9)*(3+6))"
i = 0
ONP(klasyczna, i)

```

Tym samym udało nam się zaimplementować algorytm zamiany wyrażenia w notacji infiksowej na wyrażenie w ONP.

W przykładzie dane jest wyrażenie zapisane w notacji infiksowej: `((8*9)*(3+6))`

Zamieniliśmy je na wyrażenie w notacji postfiksowej, które ma następującą postać:
`8 9 * 3 6 + *`

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

Problem 1

Napisz program, który przekształci wyrażenie arytmetyczne zapisane w notacji infiksowej (zakładamy, że wyrażenie zawiera pełne nawiasowanie, występują w nim wyłącznie liczby naturalne oraz operatory jednoznakowe) do postaci ONP. Przetestuj działanie programu dla wyrażenia arytmetycznego `(((27/5)-(5*23))+((26-4)-(23^24)))`.

Specyfikacja problemu:

Dane:

- `klasyczna` – wyrażenie arytmetyczne w notacji infiksowej z pełnym nawiasowaniem; łańcuch znaków

Wynik:

- wyrażenie arytmetyczne w ONP

> **Ważne!**
>
> W wyrażeniu zastosowaliśmy operator ^. Wykorzystujemy go jako operator potęgowania.

Twoje zadania

1. Program zamienia podane wyrażenie arytmetyczne z notacji klasycznej na ONP.

```
1 klasyczna = "(((27/5)-(5*23))+((26-4)-(23^24)))"
2
3 # Miejsce na wpisanie kodu
```

```
1
```

Słownik

notacja klasyczna (notacja infiksowa)

sposób zapisu dwuargumentowego wyrażenia arytmetycznego, w którym operator działania zostaje zapisany między operandami, np. `2 + 2`

notacja polska (notacja prefiksowa)

sposób zapisu wyrażenia arytmetycznego, w którym operator umieszczony jest przed operandami, np. `+ 2 2`

odwrotna notacja polska (ONP, notacja postfiksowa)

sposób zapisu wyrażenia arytmetycznego, w którym operator umieszczony jest za operandami, np. `2 2 +`

operand

liczba, na której wykonywane jest działanie

> **Przykład 1**
>
> Dla wyrażenia arytmetycznego:

```
27 * 5
```

liczby 27 oraz 5 to **operandy**, a znak mnożenia jest **operatorem**

pełne nawiasowanie

szczególne ustawienie nawiasów w wyrażeniu arytmetycznym w notacji infiksowej; dzięki niemu każde działanie ma jednoznacznie określone dwa operandy za pomocą ujęcia w nawiasy, a także całe działanie jest objęte nawiasem, np. `(a+(b·c))`

stos

dynamiczna struktura danych, w której elementy pobierane są z wierzchołka i odkładane na wierzchołek; struktura typu LIFO (ang. *Last-In, First-Out* – ostatni na wejściu, pierwszy na wyjściu)

Film samouczek

Polecenie 1

Napisz program, który przekonwertuje wyrażenie arytmetyczne zapisane w notacji infiksowej na jego odpowiednik w odwrotnej notacji polskiej. Zauważ, że wyrażenie nie musi zawierać pełnego nawiasowania, jak było to założone w problemie omawianym w sekcji „Przeczytaj”, zatem istotna jest tu kolejność wykonywania działań.

Przetestuj działanie programu dla wyrażenia arytmetycznego:

```
wyrazenie = '(a+b*c)/d='
```

Specyfikacja problemu:

Dane:

- `wyrazenie` – ciąg znaków, wyrażenie arytmetyczne zapisane w notacji infiksowej, gdzie dozwolonymi znakami są jednoliterowe nazwy zmiennych, operatory: dodawania (+), odejmowania (-), mnożenia (*), dzielenia (/), potęgowania (^) oraz nawiasy okrągłe: otwierające (() i zamykające ()); znak równości (=) należy interpretować jako koniec wyrażenia arytmetycznego

Wynik:

- wyrażenie arytmetyczne zamienione na ONP

Twoje zadania

1. Program przepisuje podane wyrażenie arytmetyczne zapisane w notacji infiksowej na odpowiednik w notacji postfiksowej.

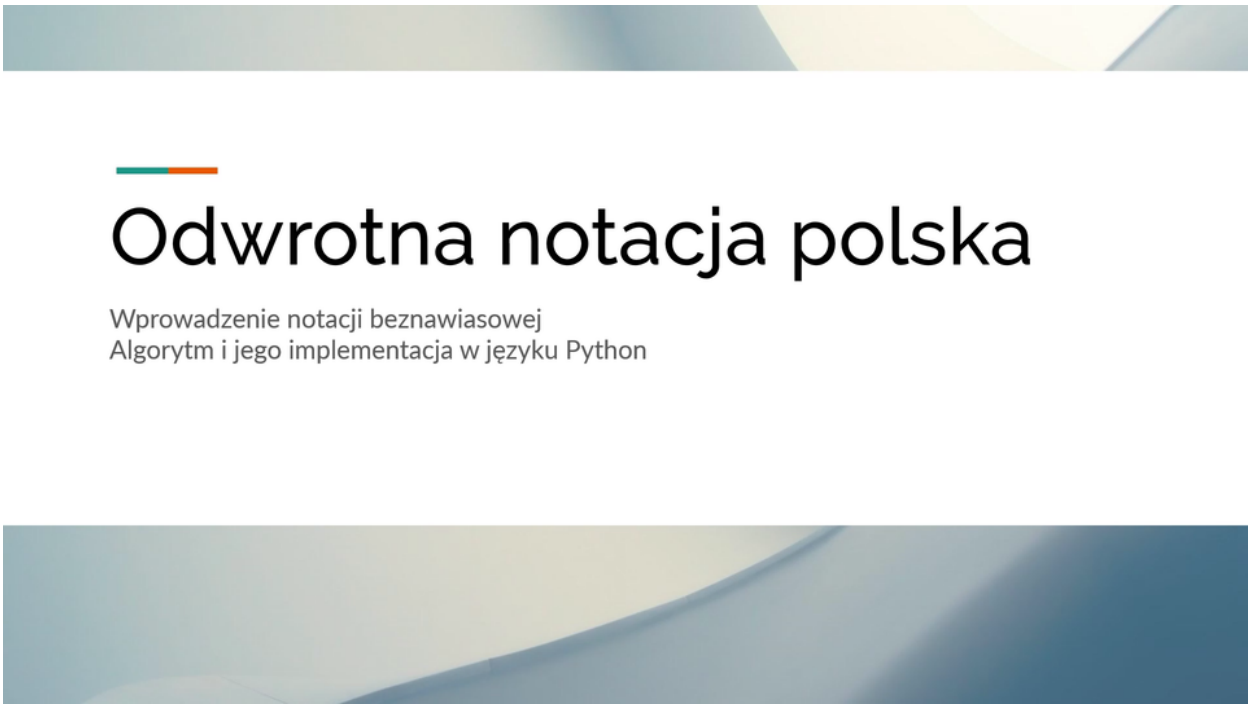
```
def pobierz_priorytet(op):
    mozliwosci = {
        '+': 1,
        '-': 1,
        '*': 2,
        '/': 2,
        '^': 3
```

```
    }
    return mozliwosci.get(op, 0)

stos = []
wyrazenie = '(a+b*c)/d='

# Wi
```

```
1
```

Polecenie 2

Porównaj swoje rozwiązanie z przedstawionym w filmie.



Film dostępny pod adresem /preview/resource/Rgf04PIBlHbg9

Film zatytułowany Odwrotna notacja polska. Wprowadzenie notacji beznawiasowej. Algorytm i jego implementacja w języku Python.

Kod programu zaprezentowanego w filmie:

Plik o rozmiarze 1.11 KB w języku polskim

Polecenie 3

Na podstawie filmu podsumuj w notatce najważniejsze informacje dotyczące konwersji wyrażeń arytmetycznych zapisanych za pomocą notacji infiksowej do postaci postfiksowej.

Polecenie 4

Uruchom symulację i przekształć wyrażenie arytmetyczne zapisane w notacji **infiksowej** na **postfiksową**. Informacje teoretyczne dotyczące tego zagadnienia znajdziesz w e-materiale Odwrotna notacja polska.

Uwaga, w symulacji nie stosujemy pełnego nawiasowania.

Symulacja 1

W poniższej symulacji interaktywnej twoim zadaniem jest samodzielne przekształcenie wyrażenia arytmetycznego zapisanego w notacji **infiksowej** na notację **postfiksową.**

Czasem by przejść dalej, konieczne będzie dokonanie wyboru z podanych opcji. W przypadku slajdów wyjaśniających zawsze możesz przejść do kolejnego slajdu (opcja `Przejdź do kolejnego slajdu`) lub wrócić do poprzedniego slajdu (opcja `Wróć do poprzedniego slajdu`), natomiast w slajdach, gdzie konieczny jest wybór kolejnego kroku rozwiązania problemu, powrót do wcześniejszego slajdu jest możliwy, natomiast przejście do kolejnego już nie.

Od trzeciego slajdu możesz również zacząć od nowa i wrócić do pierwszego slajdu (opcja `Wróć do pierwszego slajdu i zacznij od nowa`).

> **Ważne!**
>
> W wyrażeniu zastosowaliśmy operator ^. Wykorzystujemy go jako operator potęgowania.

> **Przykład 1**
>
> **W jaki sposób czytać informacje w symulacji?**
>
> 1. Wyrażenie i pobierany element: `27 * 5 + 23 - (26 / (24 ^ 4))`
> 2. Stos: +
> 3. Przekształcone wyrażenie: `27 5 * 23`
> 4. Co należy zrobić z analizowanym elementem ?
> 5. Umieść element na stosie.
>   Stos: + -
>   Wyjście: 27 5 * 23
>
>   Dodaj element do wyniku.
>   Stos: +
>   Wyjście: 27 5 * 23 -
> 6. Wróć do poprzedniego slajdu
>
>   Wróć do pierwszego slajdu i zacznij od nowa

1

Zwróć uwagę na to, że element 23 jest wyróżniony. To on jest analizowany.

2

Ten element mówi, co w danym momencie znajduje się na stosie.

3

Tak wygląda dotychczas przekształcone wyrażenie.

4

Polecenia, które należy wykonać.

5

Pierwsze wiersze obu opcji wskazują kroki, które mają zostać wykonane, natomiast kolejne pokazują, jak powinny wyglądać stos oraz aktualny wynik po wykonaniu danego kroku.

6

Kolejne opcje pozwalają na nawigację po symulacji.

Symulacja

Polecenie 5

Zapisz przykład wyrażenia arytmetycznego w notacji infiksowej. Następnie przekaż go koleżance lub koledze, a od innej osoby weź zadanie dla siebie. Przeprowadź analizę otrzymanego wyrażenia, przekształcając je do postaci postfiksowej – w razie potrzeby skorzystaj z symulacji.

Polecenie 6

Zmodyfikuj program przedstawiony w prezentacji tak, by przekształcał wyrażenie arytmetyczne zapisane w notacji postfiksowej na postać infiksową.

Uruchom swój program w środowisku lokalnym i przetestuj dla następującego wyrażenia arytmetycznego zapisanego w postaci postfiksowej:

```
8 9 * 3 6 + *
```

Zastosuj pełne nawiasowanie.

Poprawny wynik dla powyższych danych:

```
((8*9)*(3+6))
```

```
# Poniżej znajdziesz program z sekcji "Przeczytaj", który należ

def ONP(klasyczna, i):
    if klasyczna[i] != '(':

        while 0 <= (ord(klasyczna[i]) - ord("0")) <= 9:
            print(klasyczna[i], end = "")
            i += 1

        print(" ", end = "")
        return i - 1

    else:

        i = ONP(klasyczna, i + 1)
        i += 1
        operator = klasyczna[i]
        i = ONP(klasyczna, i + 1)
        i += 1

        print(operator, end = " ")
        return i

```

```
klasyczna = "((8*9)*(3+6))"
i = 0
ONP(klasyczna, i)
```

Twoje zadania

1. Program przekształca wyrażenie arytmetyczne zapisane w notacji postfiksowej na postać infiksową.

```
ONP = "8 9 * 3 6 + *"

# Miejsce na wpisanie kodu
```

```
```

Sprawdź się

Pokaż ćwiczenia: 

Ćwiczenie 1

Napisz program, który wypisze wszystkie znaki w podanym łańcuchu aż do napotkania cyfry. Przetestuj swój program dla danych:

```
ciag = "asfkbahwdkcvkjxheewid1fdsf"
```

Specyfikacja problemu:

Dane:

- `ciag` – łańcuch znaków zawierający co najmniej jedną cyfrę

Wynik:

- `napis` – łańcuch znaków będący prefiksem łańcucha `ciag`, zwierającym wszystkie znaki od początku aż do napotkania cyfry.

Twoje zadania

1. Program wypisuje wszystkie znaki z zadanego łańcucha znaków aż do napotkania cyfry.

Ćwiczenie 2

Napisz program, który wypisze wszystkie operatory arytmetyczne występujące w przekazanym wyrażeniu w postaci łańcucha znaków, bez spacji między nimi. Przetestuj swój program dla następujących danych:

```
wyrazenie = "(3/4*9)-4+8/9"
```

Specyfikacja problemu:

Dane:

- `wyrazenie` – analizowane wyrażenie matematyczne; łańcuch znaków

Wynik:

- `napis` – wynikowy ciąg operatorów arytmetycznych występujących w wyrażeniu; łańcuch znaków

Twoje zadania

1. Program dodaje operatory arytmetyczne (dodawania `+`, odejmowania `-`, mnożenia `*` oraz dzielenia `/`) z łańcucha znaków `wyrazenie` do łańucha znaków `napis`. Program wypisuje łańcuch znaków `napis`.

Ćwiczenie 3

Napisz program, który zamienia wyrażenie arytmetyczne zapisane w notacji ONP na notację klasyczną i wypisuje je w jednym wierszu. Wyrażenie w notacji klasycznej zapisane w wynikowym łańcuchu znaków powinno być wydzielone parą nawiasów okrągłych (to znaczy ma występować pełne nawiasowanie), a między operandami i operatorami powinien występować znak spacji. Zakładamy, że operatory są wyłącznie jednoznakowe.

Przetestuj swój program dla następujących danych:

```
notacja_onp = "15 9 * 25 6 + * 27 ^"
```

Specyfikacja problemu:

Dane:

- `notacja_onp` – niepusty ciąg znaków, wyrażenie arytmetyczne zapisane w notacji postfiksowej, gdzie dozwolonymi znakami są dowolne liczby naturalne oraz operatory: dodawania (+), odejmowania (–), mnożenia (*) i dzielenia (/), potęgowania (^)

Wynik:

- `klasyczna` – wyrażenie arytmetyczne zamienione na notację klasyczną z pełnym nawiasowaniem; niepusty ciąg znaków

Twoje zadania

1. Program zamienia wyrażenie arytmetyczne zapisane w notacji ONP na notację klasyczną i wypisuje je w jednym wierszu.

Dla nauczyciela

Autor: Maurycy Gast
Przedmiot: Informatyka

Temat: Odwrotna notacja polska w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, 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

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

2) wykorzystuje znane sobie algorytmy przy rozwiązywaniu i programowaniu rozwiązań następujących problemów:

d) zamiany wyrażenia na postać w odwrotnej notacji polskiej i obliczanie jego wartości na podstawie tej postaci,

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

i) struktury dynamiczne: stos, kolejka, lista (do realizacji algorytmu: ONP, symulacji problemu Flawiusza, sortowania leksykograficznego),

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):

- Wykorzystasz w praktyce sposób zapisywania wyrażeń arytmetycznych z zastosowaniem odwrotnej notacji polskiej.
- Zaimplementujesz w języku Python algorytm przekształcania wyrażenia arytmetycznego zapisanego z wykorzystaniem notacji infiksowej do postaci w ONP.
- Rozwiążesz przykładowe zadania wymagające wykazania się znajomością ONP.

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: „Odwrotna notacja polska w języku Python". Uczniowie zapoznają się z treściami w sekcji „Przeczytaj" w kontekście programowania.

Faza wstępna:

1. Nauczyciel prosi wybraną osobę o odczytanie tematu lekcji, a następnie określa cele i kryteria sukcesu.
2. Prowadzący prosi uczniów, aby zgłaszali swoje propozycje pytań do tematu. Jedna osoba może zapisywać je na tablicy. Gdy uczniowie wyczerpią swoje pomysły, a pozostały jakieś ważne kwestie do poruszenia, nauczyciel je dopowiada.

Faza realizacyjna:

1. **Praca z tekstem.** Nauczyciel ocenia, na podstawie informacji na platformie, stan przygotowania uczniów do zajęć. Jeżeli jest ono niewystarczające, prosi wybraną osobę o przedstawienie najważniejszych informacji z sekcji „Przeczytaj".
2. **Praca z multimedium.** Uczniowie zapoznają się z treścią polecenia 1 z sekcji „Film samouczek". W parach piszą program, a następnie porównują swoje rozwiązania z zaprezentowanym w filmie.
3. Uczniowie w parach rozwiązują polecenie 2 z sekcji „Film samouczek". Następnie na forum klasy omawiają swoje odpowiedzi.
4. Uczniowie w grupach wykonują polecenie 5, następnie w parach wykonują polecenie 6.
5. **Ćwiczenie umiejętności.** Nauczyciel przechodzi do sekcji „Sprawdź się". Uczniowie indywidualnie rozwiązują ćwiczenia nr 1 i 2. Osoba, która poprawnie rozwiąże zadanie jako pierwsza, wygrywa, a nauczyciel może nagrodzić ją oceną za aktywność.

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. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku Python.

Praca domowa:

1. Uczniowie wykonują polecenie 4 z sekcji „Film samouczek".
2. Uczniowie wykonują ćwiczenie 3 z sekcji „Sprawdź się".

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", „Film samouczek", „Sprawdź się" jako materiał do lekcji powtórkowej.