Python. Odczyt i zapis z/do pliku. Funkcja enumerate(). Stosy i kolejki.

Dotychczas dane wprowadzaliśmy ręcznie za pomocą klawiatury lub ”na stałe” zawieraliśmy w kodzie. Tym razem zajmiemy się podstawowym odczytywaniem i zapisywaniem zawartości do plików tekstowych.

Python posiada wbudowaną funkcję open, służącą do otwierania plików z dysku. open zwraca obiekt pliku posiadający metody i atrybuty, dzięki którym możemy dostać się do pliku i wykonywać na nim pewne operacje. Funkcja open wykorzystuje dwa argumenty, pierwszy jest nazwą pliku, drugi jest trybem odczytu lub zapisu.

Poniżej pełna lista:
„r” – Odczyt danych z pliku tekstowego.

„w” – Zapis do pliku. Jeśli plik istnieje, jego zawartość zostanie podmieniona na nową. Jeśli plik nie istnieje, zostanie stworzony.

„a” – Dopisanie danych do pliku. Jeśli plik nie istnieje, zostaje stworzony.

„r+” – Odczyt i zapis danych do pliku.

„w+” – Zapis i odczyt danych z pliku. Nowe dane zastąpią starą zawartość pliku. Jeśli plik nie istnieje, zostaje utworzony.

„a+” – Dopisywanie i odczyt danych z pliku tekstowego. Jeśli plik istnieje, nowe dane są dopisane na jego końcu. Jeśli plik nie istnieje, zostaje utworzony.

Przykładowy kod:

```
fo = open("foo.txt", "w")

fo.write("Lubię programować w Pythonie")

fo.close()
```

Funkcja write zapewnia możliwość zapisywania danych do plików w sposób bardzo podobny do odczytywania.

Otwarte pliki zajmują zasoby systemu, a inne aplikacje czasami mogą nie mieć do nich dostępu (zależy to od trybu otwarcia pliku), dlatego bardzo ważne jest zamykanie plików tak szybko, jak tylko skończymy na nich pracę. Służy do tego funkcja close.

Jeśli chcemy wyświetlić dany plik w konsoli tworzymy zmienną do której przypisujemy funkcję open z metodą read.

Poniżej przykład:

```
text = open("foo.txt").read()

print (text)
```

Poniżej inne metody zapisu oraz odczytu danych z/do pliku:
read([rozmiar]) – Odczytuje całą zawartość pliku, jeśli podamy argument, odczyta tylko wskazaną przez niego ilość znaków.

readline([rozmiar]) – Odczytuje jedną linię z pliku, jeśli podany argument, odczyta tylko wskazaną liczbę znaków z bieżącej linii. readlines() – Odczytuje wszystkie linie z pliku i zwraca jako listę.

W przypadku gdy mamy listę (lub inną sekwencję) napisów to wszystkie możemy zapisać do pliku poprzez metodę writelines:

Przykładowy kod:

```
lista = ["Kasia ", "Ania ", "Magda "]

fo = open("foo.txt", "w")

fo.writelines(lista) fo.close() text =

open("foo.txt").read() print (text)
```

Jeśli chcemy odczytać dłuższy plik tekstowy np. inwokację Pana Tadeusza należy zastosować instrukcję with oraz pętle for. With umożliwia bezpieczne zarządzanie obiektem w pamięci komputera. Funkcja split dzieli łańcuch znaków na wieloelementową listę. Separator (np ";") nie będzie występował w żadnym elemencie zwracanej listy, zostanie pominięty. Podobnie zadziała funkcja strip, która będzie pomijała białe znaki.
Przykładowy kod

```
with open("foo.txt") as plik:
   for linia in plik:
      print (linia.strip().split())
```

Funkcja enumerate()

Często, gdy mamy do czynienia z iteratorami, pojawia się również potrzeba zliczania iteracji. Python ułatwia pracę programistom, udostępniając wbudowaną funkcję enumerate () dla tego zadania.

```
x = ('apple', 'banana', 'cherry')

y = enumerate(x)
```

Funkcja enumerate() przyjmuje dwa parametry:

- iterowalny - sekwencja, iterator lub obiekty obsługujące iterację
- start (opcjonalnie) - enumerate()zaczyna liczenie od tej liczby.

Metoda Enumerate () dodaje licznik do obiektu iterowalnego i zwraca go w postaci obiektu enumerate. Ten obiekt wyliczeniowy może być następnie użyty bezpośrednio w pętlach for lub przekształcony w listę krotek przy użyciu metody list(). Jeśli chcemy sprawdzić ile linijek jest w danym tekście możemy użyć funkcji len albo enumerate.

Przykładowy kod:

```
count = len(open('foo.txt', 'r').readlines())

print (count)
```

Drugi sposób:

```
count = -1 for count,

wiersz in enumerate(open('foo.txt', 'r')):

   pass

count += 1

print (count)
```

Możemy również pobrać konkretny wiersz danego pliku za pomocą biblioteki linecache:

```
import linecache

wiersz = linecache.getline('foo.txt', 1)

wiersz1 = linecache.getline('foo.txt', 69)

wiersz2 = linecache.getline('foo.txt', 97)


print (wiersz)

print (wiersz1)

print (wiersz2)
```

Stos jest strukturą danych. Podobnie jak w przypadku list, także do stosów można dodawać elementy oraz je z nich usuwać. Jednak w odróżnieniu od list, element dodawany do stosu umieszczany jest zawsze na jego końcu i usunąć ze stosu można tylko jego ostatni element.

W przypadku dodawania elementu do stosu mówimy, że jest zapisywany na stosie lub odkładany na stos. Struktura danych tego typu, w której ostatni zapisany element jest jednocześnie pierwszym, który zostanie z niej pobrany, jest nazywana strukturą „ostatni zapisany, pierwszy obsłużony” (ang. LIFO — Last In First Out).

Poniżej przykład:

```
stos = []

stos.append(1)

stos.append(2)

stos.append(3)

stos.append(4)

print(stos)


stos.pop()

print(stos)
```

Kolejną strukturą danych jest Kolejka. Także kolejki są podobne do list; pozwalają na dodawanie i usuwanie elementów. Co więcej, kolejki przypominają nieco stosy, gdyż elementy mogą być dodawane i usuwane z nich w ściśle określonej kolejności.

Jednak w odróżnieniu od stosów, w których pierwszy element umieszczony na stosie jest jednocześnie ostatnim, który zostanie z niego pobrany, kolejki są strukturami danych „pierwszy zapisany, pierwszy obsłużony" (ang. FIFO — First In First Out) — pierwszy element zapisany w kolejce jest jednocześnie pierwszym, który zostanie z niej pobrany.

Poniżej przykład:

```
from collections import deque

kolejka = deque([])

kolejka.append(1)

kolejka.append(2)

kolejka.append(3)

kolejka.append(4)

print(kolejka)


kolejka.popleft()

print(kolejka)
```

Lista zadań:

1. Zapisz inwokację Pana Tadeusza Adama Mickiewicza w notatniku w pliku o nazwie pantadeusz.txt Następne wczytaj funkcją with open. Jak wygląda tekst? Czy można go odczytać

bez problemu? Jakie widzisz różnice w użyciu tylko i wyłącznie funkcji strip albo split? Zaproponuj rozwiązanie problemów z czytelnością tekstu.

2. W inwokacji Pana Tadeusza wyświetl 8, 12, 60,98, 104 wiersz. Czy wiersze wyświetlają się poprawnie? Sprawdź ile wierszów posiada inwokacja. Zastosuj funkcję enumerate.
3. Zaimplementuj działanie stosu za pomocą kolejki lub kolejki za pomocą stosu dla 50 wybranych liczb, 100 wybranych liczb oraz 150 liczb. Porównaj czas wykonania algorytmu. Jak zmieni się czas kiedy dane będą wczytywane za pomocą pliku? Czasy zapisz do pliku w formacie .txt
4. Napisz sortowanie kopcowe dla wprowadzonego ciągu liczb 4, 10, 11,5,73, 5, 1. Porównaj czas wykonania tego algorytmu z czasem wykonania sortowania szybkiego dla tych danych wejściowych. Zaproponuj wczytywanie liczb z poziomu plików w dowolnym formacie. Dla ilu maksymalnie wprowadzonych wartości algorytm przestanie się wykonywać? Wyniki zapisz w formacie .txt