Przygotowanie uczniów do matury z informatyki poprzez opracowanie algorytmów zadania 4.1 na maturze z 2018 r.

Roland Zimek

Nauka programowania wkroczyła do szkół na dobre. Zgodnie z Rozporządzeniem Ministra Edukacji Narodowej z dnia 14 lutego 2017 r.[1] w sprawie podstawy programowej, programowania uczą się już uczniowie w klasie pierwszej. Oczywiście w początkowym okresie nauki wykorzystywane są głównie wizualne języki programowania. Od klasy VII uczniowie powinni jednak rozpocząć programowanie w języku tekstowym. Jaki język programowania wybrać spośród dziesiątków dostępnych w internecie? To zależy już od nauczyciela. Jednak istotną wskazówką winno być przyjrzenie się liście języków programowania wymienionych w komunikacie dyrektora Centralnej Komisji Egzaminacyjnej w sprawie listy systemów operacyjnych, programów użytkowych oraz języków programowania w przypadku egzaminu maturalnego z informatyki[2]. Od roku szkolnego 2017/18 na liście tej pojawił się bardzo dynamicznie rozwijający się język Python. Obecnie jest na trzecim miejscy rankingu TIOBE najbardziej popularnych języków programowania [3]. Niewątpliwą zaletą języka Python jest łatwość programowania, intuicyjność i przejrzystość kodu, połączona z ogromnymi możliwościami.

Poziom rozszerzony na maturze z 2018 roku, zawiera zadanie 4, które należy rozwiązać tworząc odpowiedni program. Treść tego zadania jest następująca[4]:

Zadanie 4. WEGA

W ramach projektu WEGA naukowcom udało się odczytać sygnały radiowe pochodzące z przestrzeni kosmicznej. Po wstępnej obróbce zapisali je do pliku `sygnaly.txt`.

[1] http://prawo.sejm.gov.pl/isap.nsf/DocDetails.xsp?id=WDU20170000356
[2] http://www.oke.krakow.pl/inf/filedata/files/20180820%20EM%20Komunikat%20o%20egzaminie%20z%20informatyki.pdf
[3] https://www.tiobe.com/tiobe-index/
[4] https://cke.gov.pl/images/_EGZAMIN_MATURALNY_OD_2015/Arkusze_egzaminacyjne/2018/formula_od_2015/informatyka/MIN-R2_1P-182.pdf

W pliku `sygnaly.txt` znajduje się 1000 wierszy. Każdy wiersz zawiera jedno niepuste słowo złożone z wielkich liter alfabetu angielskiego. Długość jednego słowa nie przekracza 100 znaków.

Napisz program(y), który(e) da(dzą) odpowiedzi do poniższych zadań. Odpowiedzi zapisz w pliku `wyniki4.txt`, a każdą odpowiedź poprzedź numerem oznaczającym odpowiednie zadanie.

Uwaga: Plik `przyklad.txt` zawiera dane przykładowe spełniające warunki zadania. Odpowiedzi dla danych z pliku `przyklad.txt` są podane pod pytaniami.

Zadanie 4.1.

Naukowcy zauważyli, że po złączeniu dziesiątych liter co czterdziestego słowa (zaczynając od słowa czterdziestego) otrzymamy pewne przesłanie. Wypisz to przesłanie.

Uwaga: Każde co czterdzieste słowo ma co najmniej 10 znaków.

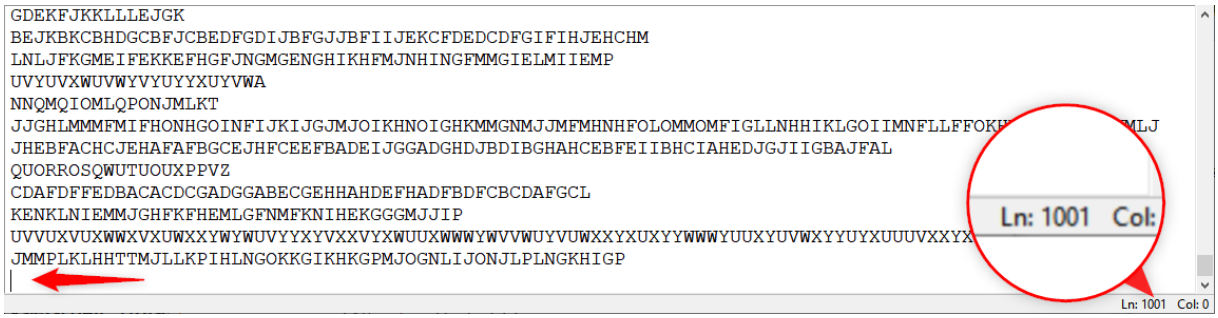
Dla danych z pliku `przyklad.txt` wynikiem jest: `NIECHCIMATURAPROSTABEDZIE`

Jak można to zadanie rozwiązać przy pomocy języka Python? Zanim to zrobię, chciałbym zwrócić uwagę na błąd, czy też nieścisłość w treści zadania. Otóż w treści występuje zapis:

W pliku `sygnaly.txt` znajduje się 1000 wierszy. Każdy wiersz zawiera jedno niepuste słowo złożone z wielkich liter alfabetu angielskiego.

który nie jest prawdziwy, co w niektórych przypadkach może powodować wystąpienie wyjątku (błędu przerywającego działanie programu) w napisanym programie.

Otwierając załączony plik `sygnaly.txt` w edytorze tekstu możemy przekonać się, że plik ten zawiera nie 1000, a 1001 wierszy. Co gorsza, ostatni 1001. wiersz jest pusty, czyli nie zawiera żadnego słowa.



Analogiczny błąd występuje także w przykładowym pliku `przyklad.txt`.

W Pythonie otworzymy plik tekstowy funkcją `open()`, której argumentami są: nazwa pliku tekstowego oraz tryb dostępu. Ponieważ zamierzamy odczytać zawartość pliku tekstowego, jako tryb dostępu podajemy `"r"`. Plik tekstowy musi być zapisany w tym samym folderze co program, w przeciwnym razie należy uwzględnić ścieżkę dostępu do niego.

Następnie należy przepisać zawartość pliku tekstowego do zmiennej. Python nie posiada ograniczenia długości tekstu zapisanego w zmiennej, co w tym przypadku jest bardzo przydatne. Użyjemy tutaj metody `.read()`, która całą zawartość pliku tekstowego zachowa pod zmienną.

Ponieważ otrzymamy bardzo długi tekst, wygodnie będzie go rozdzielić na poszczególne elementy listy. Skorzystamy w tym celu z metody `.split()`, w której jako separator wpiszemy `"\n"`, który oznacza wystąpienie znaku końca akapitu.

Program należy zakończyć metodą `.close()`, zamykającą otwarty plik.

```
plik = open("przyklad.txt","r")


tekst = plik.read()
tekst = tekst.split("\n")
print(len(tekst))


plik.close()
```

Jak możemy się przekonać, uruchomienie tego programu spowoduje wyświetlenie liczby `1001` co oznacza, że odczytanych zostało 1001 wierszy.

Jeżeli uczeń zastosował taki sposób wczytania zawartości przykładowego pliku, to w dalszej części może będzie miał problemy z poprawnością działania programu.

Zmieńmy wobec tego metodę `.read()`, na metodę `.readlines()`, która każdy wiersz z pliku tekstowego zapisze jako odrębny element listy. Dzięki temu nie będzie potrzebna już metoda `.split()`. Metoda `.readlines()` rozdziela tekst używając jako separatora `"\n"` (nie usuwa go jednak), pomijając przy tym pusty wiersz na końcu pliku. Dzięki takiemu rozwiązaniu nie napotkamy się na niespodziewane problemy z działaniem programu.

```
plik = open("przyklad.txt", "r")


tekst = plik.readlines()
print(len(tekst))
print(tekst)


plik.close()
```

Uruchomienie takiego programu spowoduje wyświetlenie liczby `1000` i tekstu na żółtym tle `Squeezed text (103 lines)`.

```
1000
Squeezed text (103 lines).
```

Ponieważ zmienna `tekst` zawiera bardzo długą listę, Python zamiast wyświetlić całość umieścił informację, że cały tekst mieści się w 103 wierszach na ekranie monitora. Możemy go zobaczyć klikając dwukrotnie tekst na żółtym tle napisu.

```
', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AAAAAAAAAI\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n
', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n
', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n
', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n
', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AZ\n', 'AAAAAAAAAE\n']
>>>
```

Zwróćmy uwagę, że lista nie zawiera ostatniego pustego wiersza, a pozostałe kończą się znakiem końca wiersza (`\n`).

Zgodnie z treścią zadania należy odczytać co 40 wiersz, rozpoczynając od 40 wiersza. Najprawdopodobniej pierwszą myślą na selekcję odpowiednich wierszy będzie użycie pętli

`for` w parze z instrukcją warunkową if. Zadaniem pętli byłoby wygenerowanie liczb z zakresu od 0 do liczby wierszy w tekście. Z kolei instrukcja warunkowa powinna sprawdzać czy są to wiersze podzielne bez reszty przez 40. Aby dodatkowo zacząć od wiersza 40. należy także wykluczyć te o mniejszych wartościach. Zapis dla takiej instrukcji mógłby wyglądać następująco:

```
for i in range(len(tekst)):
    if i>39 and not i%40:
        print(i)
```

Ponieważ jednak Python oferuje bardziej złożone możliwości określenia generowanych liczb, instrukcję warunkową `if` można zastąpić odpowiednimi argumentami generatora `range()`. Python indeksuje elementy listy od `0` (zera), musimy w takim razie w generatorze `range()` nakazać rozpoczęcie generowania liczb od wartość 39 (jest to wiersz 40, gdyż indeksujemy od 0). Zakończymy generowanie liczb, gdy osiągniemy ostatni wiersz. W trzecim argumencie możemy natomiast określić co która wartość ma zwrócić generator `range()`.

Z wybranych w ten sposób wierszy należy odczytać 10. znak (znowu pamiętajmy o indeksowaniu od `0`). Aby nie wyświetlać każdej litery w oddzielnym wierszu, zakończymy polecenie `print()` argumentem `end=""` powodującym, że zamiast nowego akapitu, będzie wstawiany pusty tekst.

```
for i in range(39, len(tekst), 40):
        print(tekst[i][9], end="")
```

Dzięki temu dowiemy się, że rozwiązaniem dla przykładowego pliku `przyklad.txt` będzie tekst:

`NIECHCIMATURAPROSTABEDZIE`

Zmieńmy w programie nazwę pliku na `sygnaly.txt`, aby odczytać właściwe przesłanie:

`ZAPISZODPOWIEDZIWPLIKUTXT`

Odczytaną odpowiedź należy zapisać w pliku poprzedzając ją numerem zadania. Musimy więc otworzyć plik `wyniki4.txt` do zapisu, używając tym razem trybu dostępu `"w"`. Nie zapomnijmy także o zamknięciu otwartego pliku metodą `.close()`.

Ponadto należy zapamiętać odczytane rozwiązanie w zmiennej, której zawartość zapiszemy do pliku metodą `.write()`.

Ostateczna postać programu (po niewielkich modyfikacjach) może wyglądać następująco:

```
plik = open("sygnaly.txt", "r")
wyniki = open("wyniki4.txt", "w")
tekst = plik.readlines()
odp = "4.1. "

for i in range(39, len(tekst), 40):
        odp += tekst[i][9]
print(odp)
wyniki.write(odp)

plik.close()
wyniki.close()
```

I mimo, że jest to postać programu dająca poprawne wyniki, warto wspomnieć o pewnej specyfice języka Python jaką są listy składane, pozwalające trochę skrócić program. Listy składane możemy wykorzystać w przypadku gdy pętla zawiera jedno polecenie do wykonania. Listy składane tworzą listę wykonując dla każdej iteracji pętli `for` zadane wyrażenie. Ogólna postać najprostszej postaci listy składanej jest następująca:

zmienna `=` `[`***wyrażenie*** `for` ***wartość*** `in` ***lista***`]`

Aby zamiast użytej w programie pętli `for`, zastosować listę składaną, należałoby wpisać:

```
odp = [tekst[i][9] for i in range(39, len(tekst), 40)]
```

W celu lepszego zobrazowania działania list składanych opiszę krok po kroku wszystkie wykonywane w tej instrukcji czynności:

[tekst[i][9] **for i in range(39, len(tekst), 40)**]	Uruchomienie pętli
[**tekst[i][9]** for i in range(39, len(tekst), 40)]	Odczytanie z wiersza znaku o indeksie 9
[tekst[i][9] for i in range(39, len(tekst), 40)**]**	Utworzenie z tych znaków listy

Jednak tak zapisane polecenie dałoby w rezultacie listę, w której każdy znak byłby oddzielnym elementem listy:

```
['Z', 'A', 'P', 'I', 'S', 'Z', 'O', 'D', 'P', 'O', 'W', 'I', 'E',
'D', 'Z', 'I', 'W', 'P', 'L', 'I', 'K', 'U', 'T', 'X', 'T']
```

Chcąc połączyć wszystkie elementy listy, możemy skorzystać z metody `.join()`. Po drobnych poprawkach program mógłby wyglądać następująco:

```
plik = open("sygnaly.txt","r")
wyniki = open("wyniki4.txt","w")

tekst = plik.readlines()
odp = "4.1. " + "".join([tekst[i][9] for i in range(39, len(tekst), 40)])
print(odp)
wyniki.write(odp)

plik.close()
wyniki.close()
```

Taka postać mimo, że jest mniej czytelna i ze względu na konieczność scalenia listy zastosowana trochę na siłę, to jednak może być interesującym ćwiczeniem, które można wykonać z bardziej zaawansowanymi uczniami.

Istnieje jednak inne, bardzo sprytne i bardzo krótkie rozwiązanie. W języku Python możemy odwołać się do dowolnego fragmentu sekwencji, podając początkowy indeks, końcowy i krok. Ogólna postać takiego odwołania jest następująca:

```
lista[początek, koniec, krok]
```

Otrzymamy w ten sposób tylko wiersze zgodne z założeniami zadania. Wystarczy teraz dla każdego z nich pobrać znak znajdujący się na indeksie 9.

```
plik = open("sygnaly.txt","r")
wyniki = open("wyniki4.txt","w")
tekst = plik.readlines()

odp = "Zad 4.1 " + "".join(i[9] for i in tekst[39::40])
print(odp)
wyniki.write(odp)

plik.close()
wyniki.close()
```