Laboratorium kryptograficzne dla licealistów 2

Projekt „Matematyka dla ciekawych świata”
Łukasz Mazurek

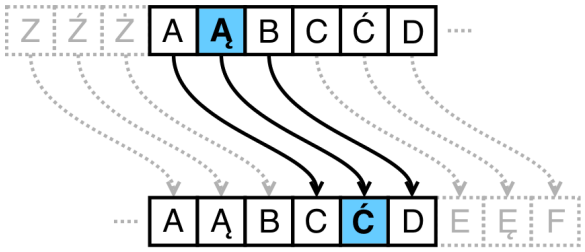
30.03.2017

1 Szyfr Cezara

Uwaga *We wszystkich zadaniach dotyczących szyfrowania (o ile nie powiedziano inaczej) zajmujemy się słowami składającymi się wyłącznie z wielkich liter 32-literowego polskiego alfabetu (AĄBCĆDEĘFGHIJKLŁMNŃOÓPRSŚTUWYZŹŻ).*

Szyfr Cezara polega na zastąpieniu każdej litery tekstu jawnego literą występującą w alfabecie o 3 pozycje później — literę A szyfrujemy jako C, Ą szyfrujemy jako Ć, itd. Gdy „skończy nam się alfabet”, zaczynamy liczyć od początku — literę Z szyfrujemy jako A, Ź jako Ą, a Ż jako B. Czyli szyfrogramem dla tekstu jawnego CEZAR jest słowo EGACT, a szyfrogram DCÓCÓ po odszyfrowaniu daje tekst jawny BANAN.

Zadanie 1 *Odszyfruj przy użyciu kartki i długopisu (nie wspomagając się komputerem) tekst jawny, który po zaszyfrowaniu szyfrem Cezara daje szyfrogram ŚŻAC.*



1.1 Funkcja `translate`

Szyfr Cezara jest przykładem tzw. szyfru podstawieniowego: każda litera alfabetu jest szyfrowana na inną literę alfabetu zgodnie z pewną *regułą*. Reguła ta może być np. następującej postaci:

- $A \longrightarrow X$
- $B \longrightarrow Y$
- $C \longrightarrow Z$

Taka reguła oznacza, że każda litera A przechodzi na X, każda litera B przechodzi na Y, a każda litera C przechodzi na Z. Regułę taką możemy krócej zapisać w postaci:

$$ABC \longrightarrow XYZ$$

Zgodnie z powyższą regułą, słowo BABA po zaszyfrowaniu przejdzie na YXYX.

Bardzo przydatną funkcją do szyfrowania podstawieniowego jest funkcja `translate` w Pythonie. Aby z niej skorzystać, należy najpierw poleceniem `str.maketrans` stworzyć regułę, według której będziemy szyfrować:

```
> regula = str.maketrans('ABC', 'XYZ')
```

Stworzona powyżej reguła oznacza, że przy szyfrowaniu litera A przejdzie na X, B na Y, natomiast C przejdzie na Z. Mając taką regułę, możemy zaszyfrować dowolne słowo przy użyciu polecenia `translate`:

```
> 'BABA'.translate(regula)
=> 'YXYX'
```

Zwróć uwagę na składnię polecenia `translate`: najpierw podaje się słowo do zaszyfrowania, następnie po kropce występuje słowo `translate`, a na końcu w nawiasie okrągłym podaje się regułę, według której chcemy zaszyfrować.

Zadanie 2 *Korzystając z poleceń* `str.maketrans` *i* `translate` *zaszyfruj szyfrem Cezara słowo:*

UZGŹHLFĆAFC

Pamiętaj, że używamy 32-literowego polskiego alfabetu: AĄBCĆDEĘFGHIJKLŁMNŃOÓPRSŚTUWYZŹŻ, a szyfr Cezara zastępuję każdą literę alfabetu literą występującą o 3 pozycje później w alfabecie.

1.2 Szyfr Cezara z kluczem k

W oryginalnym szyfrze Cezara następowało przesunięcie o 3 pozycje w alfabecie. Możemy zajmować się również *szyfrem Cezara z kluczem k*, w którym szyfrogram otrzymuje się przesuwając każdą literę tekstu o k pozycji w alfabecie. Np. dla $k = 5$ szyfr przeprowadza A na D, Ą na E, itd.

Aby zmodyfikować nasz program do szyfrowania dowolnym kluczem, musimy mieć metodę do automatycznego generowania alfabetu „przesuniętego o k pozycji w lewo". Np. dla $k = 5$ chcielibyśmy wygenerować alfabet przesunięty o 5 pozycji w lewo:

DEĘFGHIJKLŁMNŃOÓPRSŚTUWYZŹŻAĄBCĆ

Jednak to zadanie rozwiązaliśmy już na poprzednich zajęciach! Takie przesunięcie nosi nazwę *rotacji* słowa i możemy je wygenerować sklejając z sobą:

- podsłowo powstałe przez usunięcie pierwszych pięciu liter

oraz

- podsłowo złożone z tych pięciu pierwszych liter:

```
> alfabet[5:] + alfabet[:5]
=> 'DEĘFGHIJKLŁMNŃOÓPRSŚTUWYZŹŻAĄBCĆ'
```

1.3 Tworzenie własnych funkcji

Wszystkie programy, które do tej pory pisaliśmy można podzielić na dwa rodzaje:

- ciąg jednolinijkowych poleceń, które wpisujemy bezpośrednio w konsoli Pythona (prawe, czarne okienko na stronie `repl.it`),
- pętle, które wygodniej pisać w pliku z kodem źródłowym (lewym, białym okienku na stronie `repl.it`), gdyż składają się z kilku linijek

Często będziemy chcieli wielokrotnie wykorzystać raz napisany fragment kodu. W tym celu będziemy tworzyć własne *funkcje*.

Definicja funkcji ma następującą postać:

```
def <nazwa_funkcji>(<argumenty>):
  <pierwsze_polecenie>
  <drugie_polecenie>
  ...
```

Zwróć uwagę na podobieństwa definicji funkcji do pętli **for**:

- pierwsza linijka kończy się dwukropkiem,
- każda z kolejnych linijek jest *wcięta* względem pierwszej linijki.

Jako że definicje funkcji składają się zawsze z co najmniej kilku linijek, będziemy je pisać (podobnie jak pętle **for**) w **lewym okienku** interpretera (w pliku z kodem źródłowym).

Przykład Napiszmy funkcję, która wypisuje swój argument podniesiony do kwadratu:

```
def kwadrat(x):
  print(x * x)
```

```
> kwadrat(7)
49
> kwadrat(2 + 3)
25
```

Spróbuj przepisać definicję funkcji `kwadrat` do lewego okienko interpretera `repl.it` i nacisnąć przycisk „run”. Jeśli wszystko się powiedzie, od tej pory będziesz mógł *wywoływać* funkcję `kwadrat`, pisząc w prawym okienku `kwadrat(argument)`, podając dowolną liczbę (lub wyrażenie) jako `argument`.

Polecenia, takie jak wywołanie funkcji, możemy również zapisywać w lewym okienku, poniżej definicji tej funkcji. Wówczas, po naciśnięciu przycisku „run”, w prawym okienku zobaczymy efekt działania tych poleceń:

```
def dodaj(a, b):
  print(a + b)

dodaj(7, 8)
dodaj('pies', 'kot')
for liczba in [10, 20, 30]:
  dodaj(liczba, 1)
```

```
15
pieskot
11
21
31
```

Zadanie 3 *Napisz funkcję* ***`cezar(slowo, klucz)`****, która wypisze* ***`slowo`*** *zaszyfrowane szyfrem Cezara o kluczu* ***`klucz`****. Np. słowo BANAN zaszyfrowane szyfrem Cezara o kluczu 2 brzmi ĆBOBO, więc wywołanie funkcji* ***`cezar`*** *w tym przypadku powinno wyglądać następująco:*

```
> cezar('BANAN', 2)
ĆBOBO
```

2 Instrukcja warunkowa if

Często chcemy, aby program zachowywał się w różny sposób w zależności od tego, czy jakiś warunek jest spełniony, czy nie. W Pythonie (jak w większości języków programowania) służy do tego instrukcja warunkowa **if**.

Przypuśćmy, że chcemy napisać funkcję `czy_parzysta(liczba)`, która sprawdza, czy `liczba` jest parzysta, czy nie i wypisuje odpowiedni komunikat. Aby sprawdzić, czy liczba jest parzysta, możemy sprawdzić, czy reszta z dzielenia jej przez 2 jest równa 0. Zatem kod funkcji `czy_parzysta` będzie wyglądał następująco:

```
def czy_parzysta(liczba):
  if liczba % 2 == 0:
    print("Parzysta")
  else:
    print("Nieparzysta")
```

Zwróć uwagę na następujące rzeczy:

- **if** to po polsku „jeśli”, **else** to po polsku „w przeciwnym przypadku”.
- Linijki rozpoczynające się od **if** i **else** (podobnie jak linijki rozpoczynające się od **for** lub **def**) kończą się dwukropkiem.
- „Wnętrze” **if**-a i **else**-a (linijki 3 i 5) jest wcięte (podobnie jak wnętrze **for**-a lub **def**-a).
- Linijka 3 zostanie wykonana, jeśli spełniony będzie warunek z linijki 2, czyli jeśli reszta z dzielenia liczby przez 2 będzie równa 0.
- Linijka 5 zostanie wykonana, jeśli warunek z linijki 2 nie będzie spełniony, czyli jeśli reszta z dzielenia liczby przez 2 nie będzie równa 0.

W powyższym przykładzie użyliśmy konstrukcji **if**/**else** do rozróżnienia pomiędzy dwoma przypadkami. Używając komendy **elif** (skrót od **else if**) możemy stworzyć bardziej skomplikowany kod do rozróżnienia pomiędzy kilkoma różnymi przypadkami:

```
for c in 'Analfabetyzm':
  if c == 'A' or c == 'a':
    print('A lub a')
  elif c in 'xyz':
    print('x-z')
  else:
    print('Nic ciekawego')
```

```
A lub a
Nic ciekawego
A lub a
Nic ciekawego
Nic ciekawego
A lub a
Nic ciekawego
Nic ciekawego
Nic ciekawego
x-z
x-z
Nic ciekawego
```

Ten kod składa się z trzech bloków, które są wykonywane w zależności od spełnienia poszczególnych warunków: **if**, **elif**, **else**. Mamy dużą dowolność w konstruowaniu tego typu fragmentów kodu: bloków **elif** może być dowolnie wiele, blok **else** może występować jako ostatni blok, ale może też go nie być w ogóle. W powyższym przykładzie widzimy również, jak można łączyć warunki spójnikiem **and** („i”) oraz **or** („lub”) Pierwsze połączenie oznacza, że chcemy, aby spełnione były **wszystkie** z wymienionych warunków, a drugie, że chcemy aby spełniony był **co najmniej jeden** z wymienionych warunków.

Zwróć uwagę na środkowy warunek. Ma on postać „**if** `A` **in** `B`:”. Taki warunek sprawdza, czy słowo A jest podsłowem (czyli zawiera się w całości wewnątrz) słowa B. W naszym przykładzie sprawdzaliśmy, czy litera będąca wartością zmiennej `c` zawiera się wewnątrz słowa `'xyz'`, czyli czy jest jedną z liter: x, y lub z.

Zadanie 4 *Napisz funkcję* ***`wyiksuj(napis)`****, która wypisze dany* ***`napis`****, zastępując każdą małą literę polskiego alfabetu małą literą x i każdą wielką literę polskiego alfabetu wielką literą X, natomiast resztę znaków pozostawi bez zmian. Np. dla napisu* *`'FC Barcelona - Real Madryt 3:2'`* *program powinien wypisać:*

```
> wyiksuj('FC Barcelona - Real Madryt 3:2')
XX Xxxxxxxxx - Xxxx Xxxxxx 3:2
```

Wskazówka: Dla każdego znaku użyj konstrukcji **if**/**elif**/**else**, *aby rozróżnić pomiędzy trzema przypadkami: małe litery, wielkie litery, pozostałe znaki (podobnie jak w pętli przechodzącej po słowie „Analfabetyzm”).*

3 Zadania dodatkowe

Zadanie 5 *Pewne słowo zostało zaszyfrowane szyfrem Cezara o kluczu 21 i uzyskano słowo MYŹCFMFUPEN. Jakie to było słowo? Spróbuj rozwiązać to zadanie nie tworząc nowej funkcji do odszyfrowywania, a jedynie używając funkcji* `cezar`.

Zadanie 6 *Korzystając z poleceń* `str.maketrans` *i* `translate`*, napisz funkcję* `powieksz(slowo)`*, która dla danego słowa* `slowo` *składającego się z małych lub wielkich liter polskiego alfabetu, wypisze je ze wszystkimi literami zamienionymi na wielkie. Np. dla słowa McGonagall funkcja powinna zwrócić słowo MCGONAGALL:*

```
> powieksz('McGonagall')
MCGONAGALL
```

Zadanie 7 *Napisz funkcję* `bezwzgledne(lista)`*, która dla danej listy liczb wypiszę listę wartości bezwzględnych tych liczb, tj. liczby ujemne zamieni na przeciwne, a liczby nieujemne pozostawi bez zmian. Poszczególne liczby powinny być oddzielone pojedynczymi spacjami. Przykładowe użycie funkcji powinno wyglądać następująco:*

```
> bezwzgledne([5, -10, 15, 0])
5 10 15 0
```

Zadanie 8 *Rozważmy szyfr podstawieniowy według następującej reguły:*

$$AĄBCĆDEĘFGHIJKLŁMNŃOÓPRSŚTUWYZŹŻ \longrightarrow$$
$$SĆTGLOAPNEJBFHŁZIRÓUĄKĘŃŹŻDYMWCŚ$$

W tym szyfrze A przechodzi na S, Ą przechodzi na Ć, B na T, itd. Pewne polskie słowo s_0 zostało zaszyfrowane tym szyfrem i otrzymano szyfrogram s_1. Następnie słowo s_1 zostało ponownie zaszyfrowane tym samym szyfrem i otrzymano szyfrogram s_2. Następnie zaszyfrowano s_2, aby otrzymać s_3, itd. Łącznie procedurę szyfrowania powtórzono n razy i otrzymano słowo

$$ÓŚZKŃTĆLHMALÓOYE$$

Jakie było początkowe słowo s_0 i ile razy było zaszyfrowane?

4 Praca domowa nr 2

Rozwiązania zadań domowych należy przesłać do czwartku 6 kwietnia do godz. 16^{59} na adres `licealisci.pracownia@icm.edu.pl` wpisując jako temat wiadomości `Lx PD2`, gdzie `x` to numer grupy, np. `L3 PD2` dla grupy L3, itd.

Zadanie domowe 1 (1 pkt) *Pewne polskie słowo zostało zaszyfrowane szyfrem Cezara z pewnym kluczem i w wyniku otrzymano szyfrogram ZWEŚŃHĆÓUBĘŚŃK. Przy użyciu pętli* **for** *i funkcji* ***cezar*** *spróbuj rozszyfrować szyfrogram wszystkimi 31 możliwymi kluczami i znajdź słowo, które zostało zaszyfrowane.*

Zadanie domowe 2 (2 pkt) *Napisz funkcję* ***szyfruj_liste(lista)****, która wypisze listę słów zaszyfrowaną w następujący sposób: każde słowo będzie zaszyfrowane szyfrem Cezara o kluczu równym długości tego słowa. Np. lista ['ALA', 'MA', 'KOTA'] powinna zostać zaszyfrowana do ['CNC', 'ŃB', 'NSZĆ'] (pierwsze słowo jest zaszyfrowane szyfrem Cezara o kluczu 3, drugie o kluczu 2, a trzecie o kluczu 4). Poszczególne słowa powinny zostać wypisane w oddzielnych wierszach. Przykładowe użycie funkcji powinno wyglądać następująco:*

```
> szyfruj_liste(['ALA', 'MA', 'KOTA')
CNC
ŃB
NSZĆ
```

Zadanie domowe 3 (3 pkt) *Napisz funkcję* ***szyfruj_napis(napis, klucz)****, która dla danego* ***napisu*** *składającego się z małych i wielkich liter polskiego alfabetu oraz ze spacji, cyfr i znaków specjalnych, wypisze ten napis zaszyfrowany w następujący sposób: każda mała lub wielka litera alfabetu zostanie zaszyfrowana szyfrem Cezara z kluczem* ***klucz****. Każdy inny znak (spacje, cyfry, znaki specjalne) zostanie wypisany w niezmienionej postaci. Przykładowe użycie funkcji powinno wyglądać następująco:*

```
> szyfruj_napis('Polska - Włosi 8:8', 4)
Tsńwnć - Żoswł 8:8
```