

Laboratorium kryptograficzne dla gimnazjalistów 2

Projekt „Matematyka dla ciekawych świata”

Łukasz Mazurek

07.04.2016

1 Instrukcja warunkowa **if**

Pamiętacie, jak na poprzednich zajęciach rozszyfrowaliśmy szyfrogram UYZKBOMJUK zaszyfrowany szyfrem Cezara z kluczem 10? W zależności od tego, czy litera była z części alfabetu A – J, czy K – Z, do jej kodu albo dodawaliśmy 16:

```
chr(ord(c) + 16)
```

albo odejmowaliśmy 10:

```
chr(ord(c) - 10)
```

Aby zautomatyzować ten proces, możemy użyć instrukcji warunkowej **if**:

```
1 for c in 'UYZKBOMJUK':
2     if ord(c) <= ord('J'):
3         print(chr(ord(c) + 16), end = ' ')
4     else:
5         print(chr(ord(c) - 10), end = ' ')
```

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**) kończą się dwukropkiem.
- „Wnętrze” **if**-a i **else**-a (linijki 3 i 5) jest wcięte (podobnie jak wnętrze **for**-a).
- Linijka 3 zostanie wykonana, jeśli spełniony będzie warunek z linijki 2, czyli jeśli kod litery będzie nie większy od kodu litery J.
- Linijka 5 zostanie wykonana, jeśli warunek z linijki 2 nie będzie spełniony, czyli jeśli kod litery będzie większy od kodu litery J.

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':
    kod = ord(c)
    if kod == ord('A') or kod == ord('a'):
        print('A lub a')
    elif kod >= ord('x') and kod <= ord('z'):
        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.

Zadanie 1 Napisz program, który wypisze dany napis, zastępując każdą małą literę alfabetu łacińskiego małą literą *x* i każdą wielką literę alfabetu łacińskiego zastępując wielką literą *X*, natomiast resztę znaków pozostawi bez zmian. Np. dla napisu *'FC Barcelona - Real Madryt 3:2'* program powinien wypisać:

```
XX Xxxxxxxxxx - 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”).

2 Definiowanie własnych funkcji

Wszystkie programy, które do tej pory pisaliśmy, składały się z kilku linii kodu i same w sobie stanowiły pełne rozwiązanie zadania. W dłuższych programach często zdarza się, że jakiś fragment kodu powtarza się w identycznej lub nieznacznie zmienionej postaci w kilku miejscach. Wówczas warto wydzielić ten kod jako podprogram i we wszystkich miejscach, w których jest potrzebny, zamiast przepisywać/kopiować ten sam kod, używać wydzielonego podprogramu. Dzięki temu, kod całego programu będzie krótszy, a ponadto, jeżeli w przyszłości będziemy chcieli zmienić coś w tym fragmencie, konieczna będzie zmiana w tylko jednym miejscu.

Opisane wyżej „wydzielanie podprogramów” to nic innego jak definiowanie własnych funkcji. Przyjrzyjmy się funkcji **wykrzykniki**, która ma na celu wypisać napis w taki sposób, że po każdym wystąpieniu znaku znak postawi wykrzyknik. Np. dla napisu *'abrakadabra'* i znaku *'a'* funkcja ma wypisać *'a!bra!ka!da!bra!'*.

```
def wykrzykniki(napis, znak):
    for c in napis:
        print(c, end = '')
        if c == znak:
            print('!', end = '')
```

Zwróć uwagę na następujące rzeczy:

- Pierwsza linijka zaczyna się od słowa kluczowego **def**, potem występuje nazwa funkcji i jej argumenty w nawiasie okrągłym. Cała linijka kończy się dwukropkiem, a wewnątrz funkcji jest wcięcie.
- Funkcja korzysta z argumentów **napis** i **znak** tak jak ze zwykłych zmiennych.

Przepisz powyższy kod programu do lewego okienka w interpreterze `repl.it` i spróbuj go uruchomić przyciskiem „run”. Jeśli tylko nie pomyliłeś się w przepisywaniu, nie powinno się wydarzyć się nic zasługującego na uwagę. Dzieje się tak, ponieważ napisaliśmy definicję funkcji, ale jeszcze jej nie uruchomiliśmy. Aby uruchomić (wywołać) funkcję **wykrzykniki** dopisz na końcu kodu (już bez wcięcia) linijkę

```
wykrzykniki('abrakadabra', 'a')
```

i jeszcze raz uruchom cały skrypt.

Zadanie 2 Napisz funkcję *szyfruj_znak(znak, klucz)*, która jeśli **znak** jest małą lub wielką literą alfabetu łacińskiego, to zaszyfruje go szyfrem Cezara o kluczu **klucz**. W przypadku innego znaku, funkcja powinna wypisać go niezmiennego. Funkcja powinna działać dla dowolnego klucza z zakresu $\{0, 1, \dots, 25\}$. Przykładowe wywołania funkcji:

```
szyfruj_znak('Y', 3)
szyfruj_znak('b', 23)
szyfruj_znak('.', 5)
```

By .

3 Szyfr Cezara z nieznanym kluczem

Podczas zajęć już wielokrotnie udało nam się rozszyfrować szyfr Cezara. Jednak za każdym razem z góry wiedzieliśmy, jakim kluczem została zaszyfrowana wiadomość. Jak sobie poradzić w przypadku, gdy wiemy, że wiadomość została zaszyfrowana kluczem Cezara, jednak nie znamy klucza? Możemy odszyfrować wiadomość po kolei wszystkimi 25 możliwymi kluczami i „na oko” sprawdzić, w którym przypadku odszyfrowane znaki tworzą słowa istniejące w naszym języku.

Zadanie 3 Pewien napis został zaszyfrowany szyfrem Cezara o nieznanym kluczu i w wyniku otrzymano szyfrogram *Cqjucqjoaq t b*. Jak brzmiała zaszyfrowana wiadomość?

Wskazówka: napisz funkcję *szyfruj_napis(napis, klucz)*, która, korzystając z funkcji *szyfruj_znak(znak, klucz)*, zaszyfruje **napis** szyfrem Cezara o kluczu **klucz**. Następnie użyj napisanej funkcji aby wygenerować 25 wiadomości odszyfrowanych wszystkimi możliwymi kluczami i znajdź tę, która składa się ze słów istniejących w języku polskim.

4 Zadania dodatkowe

Zadanie 4 Napisz program, który wypisze ciąg liczb: 1 -2 3 -4 5 -6 7 ... 97 -98 99

Zadanie 5 Napisz program, który wypisze największą spośród liczb występujących na danej liście. Np. dla listy [5, 42, -100, 0] program powinien wypisać liczbę 42.

Zadanie 6 Napisz program, który wypisze najkrótsze spośród słów występujących na danej liście. Np. dla listy: ['Interdyscyplinarne', 'Centrum', 'Modelowania', 'Matematycznego', 'i', 'Komputerowego'] program powinien wypisać: i

5 Praca domowa nr 2

Rozwiązania zadań należy przesłać do czwartku 14 kwietnia do godz. 15⁵⁹ na adres `gimnazjalisci.pracownia@icm.edu.pl` wpisując jako temat wiadomości Gx PD2, gdzie x to numer grupy, np. G3 PD2 dla grupy G3, itd.

Zadanie domowe 1 Napisz funkcję `czy_parzysta(x)`, która wypisze słowo PARZYSTA lub NIEPARZYSTA w zależności od tego, czy wartość zmiennej `x` jest parzysta, czy nie. Np. wywołanie `czy_parzysta(7)` powinno wypisać NIEPARZYSTA, a `czy_parzysta(4)` powinno wypisać PARZYSTA.

Zadanie domowe 2 Napisz funkcję `powieksz(slowo)`, która wypisze słowo, zamieniając wszystkie małe litery na wielkie, a wielkie pozostawiając bez zmian. Np. wywołanie `powieksz('LaTeX')` powinno wypisać słowo LATEX. Możesz założyć, że słowo będzie się składało wyłącznie z małych i wielkich liter alfabetu łacińskiego.

Zadanie domowe 3 Napisz funkcję `szyfruj_napis2(napis, klucz1, klucz2)`, która będzie szyfrować kolejne znaki zmiennej `napis` przy użyciu funkcji `szyfruj_znak` (tej samej, co w zadaniu 3 z zajęć) na przemian kluczami `klucz1` i `klucz2`: pierwszy znak kluczem `klucz1`, drugi znak kluczem `klucz2`, trzeci znak kluczem `klucz1`, itd. (numer znaku liczymy względem całego napisu, a nie poszczególnych słów). Np. wywołanie `szyfruj_napis2('Ala ma kota.', 1, 3)` powinno wypisać: Bob nd npwb.

Pewne zdanie w języku polskim zostało zaszyfrowane funkcją `szyfruj_napis2` przy użyciu pewnych dwóch kluczy i w efekcie otrzymano szyfrogram `Oror q nbqh, bbeufz ygra`. Jak brzmiało zdanie przed zaszyfrowaniem?

Wskazówka: Moglibyśmy próbować rozwiązać to zadanie tak, jak zadanie 3 z zajęć: wygenerować wyniki dla wszystkich 25 możliwych wartości pierwszego klucza i dla wszystkich 25 możliwych wartości drugiego klucza, a następnie „na oko” ocenić, który z otrzymanych wyników jest poprawnym tekstem w języku polskim. Niestety, w ten sposób otrzymalibyśmy aż $25 \cdot 25 = 625$ różnych tekstów do przejrzania. Zauważmy jednak, że w naszym tekście występuje słowo jednoliterowe, a w języku polskim jest bardzo mało słów literowych. Dlatego z góry możemy odrzucić większość par kluczy i zostawić od przejrzania tylko te, które generują poprawne słowo jednoliterowe z litery `q`.