Programowanie w języku Python
Słowniki i funkcje

Rafał Grzeszczuk - grzeszcz@agh.edu.pl

3 marca 2020

Celem tego laboratorium jest zapoznanie z bardziej złożonymi strukturami danych - zbiorami par klucz-wartość (słownikami), a także przedstawienie zagadnienia funkcji. Na końcu umieszczone są zadania podsumowujące, które warto dokończyć w domu w celu utrwalenia wiadomości. Staraj się nie kopiować przykładów, zwłaszcza jeżeli nie masz doświadczenia w programowaniu. Jeżeli napotkasz trudności, nie wahaj się – poproś prowadzącego o pomoc.

Słowniki

Słownik to wbudowana w Pythona struktura danych. Działa na nieco podobnej zasadzie jak lista - która, jak pamiętamy, każdej wartości przyporządkowywały kolejny indeks – liczbę naturalną. Indeks stanowił o kolejności elementów listy i pozwalał na jednoznaczną identyfikację każdego z nich. W przypadku słowników indeksami nie muszą być liczby – mogą być nimi dowolne ciągi tekstowe. Dane w słowniku są przechowywane w sposób nieuporządkowany. Słowniki w Pythonie możemy deklarować za pomocą nawiasów wąsatych \{\} – przykładowy słownik znajdziesz na listingu poniżej.

Tak naprawdę indeksem może być dowolny typ danych, o ile jest *haszowalny*. O haszowaniu przeczytasz więcej w materiałach dodatkowych.

```
myEmptyDictionary = {}       #pusty słownik
myDictionary = {"klucz 1": "wartosc1",
                "klucz2": 123,
                17: None
               }
```

Dostęp do danych słownika odbywa się w podobny sposób jak przy listach – za pomocą operatora nawiasu kwadratowego:

```
myDictionary = {"klucz 1": "wartosc1", "klucz2": 123}
print(myDictionary["klucz 1"])
print(myDictionary["nie ma"])        #błąd wykonania
myDictionary["nie ma"] = "a jednak jest"
print(myDictionary["nie ma"])        #teraz zadziała
```

Słowniki mogą być bardzo pomocne, gdy chcemy móc przechowywać dane o konkretnej strukturze – na przykład policzyć wystepowanie wszystkich słów w tekście czy prowadzić statystyki częstości wystepowania liter (co bywa przydatne przy łamaniu szyfrów...), ale także w wygodny sposób przechowywać informacje o osobach (imię, nazwisko, adres). Zwróć uwagę, że wartości można kluczom przypisywać dynamicznie, a klucze nie muszą wcześniej istnieć. To bardzo wygodne,

gdyż nie musimy martwić się formatem danych przed ich wczytaniem do programu.

Ćwiczenia

1. Napiz program, który korzystając ze słownika zrealizuje ”słownik” polsko-angielski. Powinien znać kilka przykładowych wyrazów i tłumaczyć w jedną stronę.
2. Napisz program, który wczyta od użytkownika tekst, a następnie stworzy słownik, w którym kluczami będą poszczególne słowa, zaś wartościami - długość tych słów.
3. Napisz program, który policzy w ilu wyrazach występują poszczególne spółgłoski: aeiouy, a wynik umieści w słowniku.

Ze słownika możemy także wydobyć listę jego kluczy oraz listę jego wartości. Służą do tego funkcje `keys()` i `values()`:

```
1 myDictionary = {"klucz 1": "wartosc1", "klucz2": 123}
2 print(myDictionary.keys())
3 print(myDictionary.values())
```

Pozwala to na przeiterowanie się po wszystkich elementach słownika według listy kluczy:

```
1 for key in myDictionary.keys():
2     print(myDictionary[key])
```

lub po prostu wypisanie wszystkich jego wartości:

```
1 for val in myDictionary.values():
2     print(val)
```

Nie są to jednak rozwiązania najbardziej eleganckie. Jeżeli potrzebujemy dostać zarówno klucz jak i wartość (np. nazwisko i wiek), możemy w ramach jednej pętli `for` wydobyć jednocześnie zarówno klucz, jak i wartość za pomocą funkcji `items`:

```
1 for key, value in myDictionary.items():
2     print(key, value)
```

W każdej iteracji powyższej pętli zmienna `key` przyjmie pewną wartość klucza, a zmienna `value` odpowiadającą mu wartość.

Czasami przy dostępie do słownika nie mamy pewności, czy dany klucz się w nim znajduje. Niestety, odwołanie do nieistniejącego klucza za pomocą nawiasów kwadratowych skutkuje błędem. Możemy co prawda zweryfikować najpierw, czy dana wartość istnieje za pomocą `if`-a

```
if "a" in myDict.keys():
    print(myDict["a"])
else:
    print("brak klucza!")
```

...jednak zajmuje to sporo miejsca i wymaga przejrzenia całej listy kluczy (patrz margines). Na szczęście Python dostarcza metodę "bezpiecznego" dostępu do wartości w słowniku: `get("klucz", "wartosc domyslna")`:

```
print(myDict.get("a", "brak klucza!"))
```

Próbuje ona uzyskać wartość dla podanego klucza, ale gdy tego klucza w słowniku nie ma – zwraca wartość domyślną (jej podanie jest nieobowiązkowe; może być stringiem, liczbą, listą lub czymkolwiek innym)

Ćwiczenia

1. Napisz program, który policzy częstość występowania poszczególnych wyrazów w tekście i zapisze wynik w słowniku.
2. (*) Otrzymaliśmy z systemu kontroli wjazdów listę następującej postaci: `["KR121FG,Jan Kowalski", "KR321FG,Adam Nowak"]` Podaj nazwiska pracowników, którzy posiadają więcej niż jeden samochód. Pełną listę uzyskasz po pobraniu z moodle pliku lab3.py do katalogu, w którym pracujesz. Następnie, w pierwszej linijce zaimportuj ten plik:

   ```
   from lab3 import pojazdy
   ```

   od tej pory możesz korzystać ze zmiennej pojazdy z gotową listą.
3. (**) Korzystając z analizy częstotliwości występowania znaków spróbuj bez wykonywania ataku *brute-force* złamać szyfr ROT (znany też jako szyfr Cezara). Podpowiedź: Jaka jest najczęściej występująca litera w języku polskim, a jaka jest druga taka litera? Jakie są najczęściej występujące litery w szyfrogramie? Czy na tej podstawie jesteś w stanie obliczyć klucz?

Funkcje

Z funkcji tak naprawdę korzystamy od pierwszych zajęć. `print()` jest funkcją, `int()` to też funkcja. Do tej pory były dla nas swego rodzaju czarnymi skrzynkami – robiły swoje, a my nie interesowaliśmy się zasadą ich działania. Dziś to się zmieni. Po co funkcje? Po pierwsze: upraszczają kod. Zmniejszają liczbę powtórzeń tego samego i pozwalają lepiej go zorganizować. Po drugie: niekiedy dają lepszą kontrolę nad wykonywaniem programu i pozwalają obejść się bez

zmiennych-flag lub zmniejszają ich ilość. Po trzecie: ułatwiają współpracę: możemy umówić się z innym programistą, że zaimplementuje pewną część kodu i dostarczy go w formie funkcji przyjmującej uzgodnione uprzednio argumenty.

A czym są funkcje? Matematyczna definicja mówi o przyporządkowaniu każdemu elementowi z dziedziny dokładnie jednego elementu ze zbioru wartości – zawsze tego samego, bo przecież x^2 nie może dzisiaj dawać 4, a jutro 5. Jednak ani ta definicja prosta, ani w programowaniu szczególnie przydatna – mamy przecież takie funkcje jak random. Dla nas funkcję stanowić będą trzy elementy:

1. Wejście, czyli argumenty
2. Przepis, czyli co funkcja ma zrobić
3. Wyjście, czyli wartość zwracana

Aby **zdefiniować** funkcję, czyli określić powyższe trzy elementy, skorzystamy ze słowa kluczowego `def`.

```
1 def funkcja(argument1, argument2):
2     zmienna_wewnetrzna = argument1 + argument2
3     return zmienna_wewnetrzna + 1
```

Podawanie argumentów nie jest obowiązkowe, a ich liczba nie jest ograniczona – jednak należy je przy wywołaniu podać. Również wartość zwracana (podawana po słowie kluczowym `return`) nie jest obowiązkowa – gdy jej brak, to funkcja po **wywołaniu** zwróci wartość `None`. Wywołanie powyższej funkcji przedstawia się następująco:

Możemy też stosować w deklaracjach funkcji argumenty nieobowiązkowe, z wartością domyślną. Wtedy nagłówek wyglądałby tak: `def f(a1, a2=0)`. Jeżeli `a2` nie zostanie podane przy wywołaniu funkcji, przyjmie wartość 0. Przykładem takiego argumentu jest np. `end` z funkcji print.

```
1 zwrocona_wartosc = funkcja(3, 6)
```

Należy pamiętać, że to co zadeklarowane w funkcji – zostaje w funkcji. Dlatego poza deklaracją (blok kodu zależny od `def`) nie wolno nam używać zmiennych tam zadeklarowanych. W funkcjach można oczywiście korzystać ze wszystkich mechanizmów języka (nie można jednak korzystać z `return` poza funkcją!) – na przykład z pętli czy instrukcji warunkowych. Poniżej można znaleźć funkcję, która realizuje wyszukiwanie nazwiska w liście nazwisk:

```
1 def szukajNazwiska(szukane, listaNazwisk):
2     for nazwisko in listaNazwisk:
3         if nazwisko == szukane:
4             return True
5     return False
```

W tym wypadku nie musimy korzystać z flagi `czyZnaleziono`, jak miało to miejsce w ”bezfunkcyjnej” wersji tego zadania. Dzieje się tak dlatego, że `return` natychmiast wychodzi z funkcji - więc jeżeli tam dotrzemy, to na pewno nie dotrzemy do drugiego returna.

Słów kilka o argumentach

Jeżeli przekazujemy do funkcji zmienne, to może ona przyjąć je na dwa sposoby: skopiować wartość albo otrzymać wskazanie do miejsca w pamięci, gdzie znajduje się dana zmienna. W tym pierwszym przypadku modyfikacja wartości zmiennych w funkcji nie wpływa na ich wartość poza nią:

```
1 def f1(arg1):
2     arg1 = None
3 x = 123
4 f1(x)
5 print(x)
```

W tym drugim natomiast – mamy do czynienia z *referencją*.

```
1 def f1(arg1):
2     arg1.clear() # czysci liste
3 x = [123,123]
4 f1(x)
5 print(x)
```

Z tego właśnie powodu niektóre funkcje operujące na listach nie zwracają wartości – mówiliśmy, że wykonywały operacje "w miejscu", jak np. `random.shuffle()`. Na razie możesz przyjąć, że typy proste (`int`, `str`, `float`...)przekazuje się przez wartość, a złożone (listy, słowniki) – przez referencję

Ćwiczenia

1. Napisz funkcję, która losuje znak z podanego ciągu i zwraca go.
2. Przerób generator haseł tak, aby wykorzystywał funkcje (podpowiedź: w tym wypadku optymalna ilość to 2 własne deklaracje)
3. (*) Napisz funkcję, która zamienia losowo litery w wyrazie - poza pierwszą i ostatnią. Następnie wywołaj tę funkcję na każdym wyrazie w dowolnym, dłuższym fragmencie tekstu i oceń, czy da się zrozumieć jego treść. Podpowiedź: możesz skorzystać z funkcji `random.shuffle()` – pamiętając, że funkcja ta nie zwraca wartości, a operacje wykonuje "w miejscu".

Zadania podsumowujące

1. Napisz program, który zautomatyzuje łamanie szyfru Cezara na podstawie analizy innego, niezaszyfrowanego tekstu. Program powinien "nauczyć się" statystyk występowania liter, a następnie użyć tej wiedzy do obliczenia wartości klucza. Program powinien wykorzystywać funkcje.

Informacje końcowe

Koniecznie poćwicz w domu przerobiony do tej pory materiał. Jeśli nie pracujesz na swoim koncie imiennym, koniecznie zapisz wyniki swojej pracy – nie będą one dostępne na innych komputerach w sieci. Możesz to zrobić np. pakując pliki źródłowe i wysyłając je na platformę moodle.

Materiały dodatkowe

- Tablice mieszające, czyli o haszowaniu słów kilka https://eduinf.waw.pl/inf/alg/001_search/0067e.php
- Funkcja skrótu w kryptografii – czym są "hasze haseł", które czasem wyciekają? https://hackernoon.com/cryptographic-hashing-c25da23609c3
- Globalna `__main__` – może przydać się przy pisaniu większych projektów https://docs.python.org/3/library/__main__.html
- Bardzo proste zadania z algorytmiki https://projecteuler.net

Serwis I LO w Tarnowie to bardzo dobre miejsce do rozpoczęcia przygody z algorytmiką

Notatki