1. (5pt) Napisz program, który dla danego pliku tekstowego wróci następujące informacje: liczba bajtów, liczbę słów, liczbę linii oraz maksymalną długość linii.

```
$ python wordcount.py tekst.txt
liczba bajtów: 4956
liczba słów: 1018
liczba linii: 528
maksymalna długość linii: 67
```

9. (5pt) Napisz program, który koduje oraz dekoduje dowolny plik binarny w kodzie Base64. Kod Base64 koduje w taki sposób, że każde kolejne 6 bitów z pliku kodowane jest znakiem ASCII przedstawionych w poniższej tabeli:

```
        tablica =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
```

Na przykład: słowo "Python" kodujemy jako "UHl0aG9u"

```
|         P      |       y       |       t       |
+ - - - - - - - -|- - - - - - - -|- - - - - - - -| ...
: 0 1 0 1 0 0 0 0|0 1 1 1 1 0 0 1|0 1 1 1 0 1 0 0| ...
+ - - - - - -|- -|- - - -|- - - - - - -|  ...
:     U      |     H     |      l    |  ...

Kodowanie:
  $ python base64.py --encode plik.bin plik-enc.txt
Dekodowanie:
  $ python base64.py --decode plik-enc.txt plik.bin
```

**Uwaga:** Proszę napisać podstawowe funkcje kodujące i dekodujące bez korzystania np. z biblioteki **base64** (lub innych realizujących to zadanie).

13. (5pt) Napisz program, który zamienia wszystkie nazwy w danym katalogu oraz wszystkich podkatalogach na małe litery. Jako parametr podajemy katalog (zobacz moduł os).

```
        $ python tolower.py ./
```

17. (10pt) Napisz program, który w danym katalogu znajdzie wszystkie pliki, które powtarzają się więcej niż raz (zobacz os, help(os.walk)). Weź pod uwagę, że pliki mogą mieć różne nazwy, ale tą samą zawartość, dlatego przyjmujemy, że dwa pliki są takie same, jeśli mają taką samą wielkość oraz taką samą wartość funkcji haszującej. Przykład użycia funkcji haszujących

```
        $ python
        >>> import hashlib
        >>> hashlib.md5('python'.encode()).hexdigest()
        '23eeeb4347bdd26bfc6b7ee9a3b755dd'
```

```
        >>> hashlib.sha512('python'.encode()).hexdigest()
        'ecc579811643b170cbd88fd0d0e323d1e1acc7cef8f73483a70abea01a89
...
```

Na wyjściu program wyświetla listę wszystkich plików, które się powtarzają (nazwy plików wraz ze ścieżką)

```
$ python repchecker.py ./
---------------------------------
./plik1.txt
./katalog1/plik5.txt
./katalog3/plik1.dat
---------------------------------
./plik3.txt
./katalog2/plik2.txt
---------------------------------
```

26. (15pt)* Napisz program, który szyfruje i deszyfruje dany plik algorytm RSA. Do sprawdzania, czy dana liczba jest pierwsza wykorzystaj dowolny test pierwszości np. test Millera-Rabina. W programie powinna być możliwość generowania kluczy do szyfrowania i deszyfrowania plików (**key.pub** - klucz publiczny, **key.prv** - klucz prywatny). Np.

```

        $ python rsa.py --gen-keys 128
        $ ls
        rsa.py key.pub key.prv

```

Klucze mogą być dowolnie długie np. **bits=128** bitowe, czyli **p** i **q** mają wtedy długość **int(log(2)/log(10)*(bits))** cyfr. Przy uruchomianiu programu z parametrem --encrypt oraz ciągiem znaków do zakodowania wynik zostaje wyświetlony na standardowym wyjściu (kodowanie korzysta z klucza key.pub, jeśli kluczy nie ma to wyświetlony zostaje błąd):

```
$ python rsa.py --encrypt Python
21437302530112407657289772777280768429
```

Dekodowanie odbywa się podobnie (dekodowanie korzysta z klucza key.prv):

```
$ python rsa.py --decrypt 21437302530112407657289772777280768429
Python
```