Nalezení nejkratšího a nejdelšího slova v textu.

Zadání: Pro vstupní text zahrnující písmena "A-Ž", "a-ž", číslovky "0-9", speciální znaky ",.?!;" oddělené mezerami, nalezněte nejkratší a nejdelší slovo v textu, určete počet jejich znaků. Pokud takových slov bude více, vypište je všechny v pořadí, v jakém se v textu vyskytují. Pokud budou v textu nepodporované znaky, ignorujete je.

Problém netřeba příliš dlouho vysvětlovat – porovnává se délka slov v textu. Praktické využití je nízké – snad jen možná pro nějakou kontrolu slovní zásoby, aby slova v textu nebyla příliš dlouhá - ale jde o dobrý trénink práce se stringy. Také je možné použít jeho prvky pro jiné programy vyžadující parsing textu podle slov.

Zvolený algoritmus vymaže z věty diakritiku, a poté ji podle mezer převede na list jejích slov. Následně pak projde tento list, a u každého slova určí jestli je delší (kratší) než aktuální nejdelší (nejkratší) – pokud ano, pak jej umístí do bufferu, a tam jím nahradí aktuální(ho) držitele. Jakmile dojde na konec textu, pak vypíše obsah tohoto bufferu. Tento algoritmus jsem si vybral jednoduše proto, že jsem podobnou úlohu řešil na gymnáziu, i když kód je napsaný znovu, a vylepšený o dodatečné funkce pro zvládání nestandardních vstupů a práci se soubory.

Třída *filewc* jednoduše zvládá vstup z a výstup do textového souboru – v základu je pojmenovaný *text.txt*. Metoda téhož jména načte pomocí *readlines* jeho obsah, použije ho v třídě *wordlen*, a pak na konec tohoto souboru vypíše výsledek obou jejích metod.

Třída *wordlen* využívá dvojici metod (*minl, maxl*) s téměř identickým kódem. Zde je varianta pro nejdelší slovo (*maxl*):

- pomocí vlastní funkce *strcheck* ověří, jestli je zadaná věta validní vstup – tedy že se jedná o typ *str*, a že má nenulovou délku.

- pomocí funkce *translate* odstraní speciální znaky, funkce *split* rozdělí větu na list slov, a proměnná *maxl* je nastavená na délku prvního slova

- *For* smyčka postupně prochází slova, a porovnává jejich délku *len* s proměnnou *maxl* (nastavenou na počátku smyčky na hodnotu délky prvního slova)

- pokud má aktuální slovo délku přesně rovnou *maxl*, pak se přidá do listu *long*

- pokud je *delší* než *maxl*, pak se obsah listu *long* nahradí tímto slovem, a *maxl* se nastaví na jeho délku
Např. pokud je list [délka, šířka], tedy délka maxl je 5, a program je u slova „maximum", pak list po operaci je [maximum], a maxl je rovno 7

- jakmile skončí smyčka, podprogram vrátí list *long*, konvertovaný pomocí *join* na string oddělený čárkami

Zde je kompletní kódový blok pro ukázku:

```
def maxl(self):
    words = ''.join(char for char in self.text if char.isalnum() or char.isspace())  # cleans up string
    words = words.split()  # changes the string into an array of words
    maxl = len(words[0])  # sets initial value
    long = []
    for i in words: # goes through words
        if len(i) > maxl: #if the word is the new longest
            maxl = len(i) #sets new length to beat
            long = [i] #overwrites current list
        elif len(i) == maxl: # if of equal length, adds to the list
            long.append(i)
    return ", ".join(long)
```

- varianta pro nejkratší slovo využívá proměnnou *minl*, list *short*, a porovnává jestli je délka slova *kratší* než *minl*, ale jinak funguje téměř identicky

Toto zadání bylo relativně přímočaré, primární změna v průběhu bylo rozdělení na dva podprogramy, místo toho aby smyčka zároveň hledala obě (toto rozhodnutí umožňuje hledat jen jedno z nich, pokud je třeba), a modifikace výstupu tak, že místo toho aby buffer bylo jedno slovo a pokaždé bylo přepočítáváno, tak je buffer list pro více slov, a délka slova je separátní proměnná. Také jsem upravil metodu čištění stringu od nežádoucích znaků z funkce *translate* na modifikovaný *join*.

Testovací data mohou být třeba tato:

Vstup: *Ahoj, tak jak se vede?*

Výstup: *nejkratší: se ; nejdelší: Ahoj, vede*

Nebo tato:

„Představa o nutnosti chytrácké diplomacie se přežila; lidé počínají chápat, že lež je hloupá a zbytečně komplikuje a zdržuje jednání. Pravda je ve všem, i v politice, nejpraktičtější.“

nejkratší: o, a, a, i, v; nejdelší: nejpraktičtější

Šifrování a dešifrování textu: Vigenerova šifra.

Zadání: *Vstupní text zadaný uživatelem zahrnující písmena "A-Z" zašifrujte a dešifrujte Vignerovou šifrou s předem zvoleným klíčem. Ve výsledném textu ponechte mezery a diakritická znaménka.*

Vigenèrova šifra je symetrická polyalfabetická substituční šifra, která text šifruje pomocí série různých Caesarových šifer v závislosti na písmenech klíče.

- *Wikipedia*

To znamená, že se dané písmeno textu posune v abecedě o tolik pozic, kolikáté je v abecedě odpovídající písmeno klíče tvořeného jiným textem. Je odolná proti frekvenční analýze (protože jakýkoliv znak z otevřeného textu lze zašifrovat na libovolný jiný znak), a i proti řešení hrubou silou (díky vysokému počtu klíčů – např. S klíči o 13 písmenech má 10^{18} možných klíčů). Ve své době byla dokonce považována za nerozluštitelnou – což už ovšem dnes neplatí, ale stále je při každodenním použití velmi silná. Jednou z jejích slabin ovšem je, že je velmi složitá na vypracování. Počítačový program ale tuto práci zvládne v rámci zlomku vteřiny. Což se hodí pro spisovatele, milence, a jiné kteří chtějí skrýt před někým informace.

Zvolený algoritmus víceméně kopíruje konvenční řešení Vigenerovy šifry. Písmeno po písmenu projde celou zprávu, každé převede na jeho pozici v abecedě, posune o abecední pozici písmena v klíči, a pak konvertuje zpět na písmeno finální zprávy. V případě že je klíč kratší než zpráva jej algoritmus opakuje.

Třída *filein* zvládá vstup z *txt* souboru – v základu *cipher.txt*. Požadovaný formát vstupu je napřed text, pak klíč, a nakonec klíčové slovo *crypt* nebo *decrypt* (nebo číselný vstup 1/-1) – všechny prvky oddělené středníkem. Pokud vstup není v tomto formátu, pak jej program ignoruje. Program zároveň předpokládá, že na každém řádku je nová šifra, a tedy prochází zdrojový soubor řádek po řádku. Oba textové vstupy musí také používat anglickou abecedu, tj. bez háčků a čárek. Interpunkci, mezery a velká písmena ale program přepisuje přesně.

Hlavní je třída *Data*. Klíčová metoda *vignere* je multifunkční, tím že to jestli je zpráva (hodnota *text*) zašifrovaná nebo dešifrovaná záleží na směru posunu, určeném podle zadaného koeficientu *ed*, kterým se vynásobí index klíče (hodnota *key*) tak, aby byl kladný nebo záporný. Funguje takto:

- pomocí vlastní funkce *strcheck* ověří, jestli jsou zpráva a klíč validní vstup – tedy typ *str s* nenulovou délkou.

- jednoduchý try blok ověří, jestli má koeficient ed správnou hodnotu - tedy +1 nebo -1 (metoda také přijímá i vstup *crypt* nebo *decrypt*, které si v tomto bloku zkonvertuje na +-1). Pokud ne, pak vypíše do konzole chybovou hlášku

- vlastní funkce *chartopos* a *postochar* konvertují písmena na jejich index v abecedě a obráceně, což umožňuje k sobě „přičítat“ písmena (další zmínky o „přičítání“ písmen ve skutečnosti znamenají přičítání čísel reprezentující jejich pozice v abecedě)

- pokud má klíč méně písmen než zpráva, pak program určí kolikrát se vejde kód do zprávy, poté tolikrát zopakuje string klíče (a jednou navíc pro případ neúplného dělení). Toto je relativně snadné díky schopnosti Pythonu kopírovat string identicky jako kterýkoliv jiný datový typ.
Např. z klíče „key“ se stane „keykeykeykey“ pro zprávu o 7 písmenech

```
if len(self.key) < len(self.text):  # modifies the key, if it is shorter than the text
    self.key *= (len(self.text) // len(self.key) + 1)
```

- *For* smyčka postupně prochází znaky zprávy, každý z nich konvertovaný na jeho pozici v abecedě

- pokud je znak velké písmeno, pak se toto označí nastavením proměnné *up* na True, a s písmenem se nadále pracuje, jako by bylo malé

- pokud znak není písmeno (určeno pomocí knihovny *string* a jejího seznamu *ascii_letters*), pak jej jen uloží do výstupu

- pokud je znak písmeno, pak se k němu přičte písmeno na odpovídající pozici v klíči
Např. „a“ (1. písmeno) ve zprávě plus „f“ (6. písmeno) na odpovídající pozici klíče dají výsledek „g“ (7. písmeno)

```
sol = chartopos(let)
if let in ascii_letters:  # encrypts current letter according to the key
    sol += chartopos(self.key[pos]) * self.ed
```

- ve všech variantách, pokud je číselný výsledek větší než 26 nebo menší než 0, pak se od něj odečte nebo k němu přičte 26, aby písmeno po Z začínalo počítat zase od A
Např. „x“ (24. písm.) plus „f“ (6. písm.) dají jako výsledek „d“ (30 - 26 = 4. písmeno)

- po přičtení relevantního písmena klíče se výsledek konvertuje zpět na písmeno – malé pokud je u*p* False, velké pokud je *up* True, a toto písmeno se přidá do stringu výsledné věty

- pokud je použitá metoda *fileout*, pak program zapíše v pořadí výchozí text, klíč a výsledný text do souboru *output.txt*, spolu s poznámkami o tom, co je co – tato

metoda zároveň nezávisle ověřuje správnou hodnotu koeficientu *ed*, a přeskočí pokus o šifrování v případě že je špatně. Pozor, tento soubor se při inicializaci třídy *filein* kompletně vymaže!

Toto zadání bylo složitější, a vyžadovalo několik předělávek. Původně jsem také chtěl, aby bylo řešení napsáno do původního souboru, ale kvůli složitosti implementace jsem se rozhodl použít oddělený výstupní soubor. Bylo také zapotřebí přidat funkci pro přenesení mezer a diakritiky, a kompletně přepracovat jak zvládá klíče kratší než zpráva – originální implementace se snažila zjistit jakému písmenu v klíči odpovídá písmeno zprávy dynamicky, asi takto:

```
elif len(key) < len(text):  # řeší wraparound pokud je kód kratší než zpráva
    cut = (len(text) // len(key))
    print(cut, pos, pos // cut, pos % cut)
    print(key[pos % cut])
    sol = x + chartopos(key[pos % cut]) * ed
```

To ovšem nefungovalo, a pokusy o upravení se setkaly s neúspěchem. Bylo tedy implementováno nové řešení, které upraví klíč do dostatečně dlouhé podoby předem – což je řešení které sice nepovažuji za až tak elegantní, ale funguje.

Původně řešení využívalo separátní podprogramy pro zašifrování a dešifrování, ale pro jednoduchost byl kód přepracován tak, aby šifrovací podprogram dokázal posuny „dopředu“ i „dozadu“. Například pokud je zadání:

We are attacking soon at noon, keys, crypt

pak program zkopíruje text a klíč do výsledného souboru spolu s popisky o co se jedná, a pod něj napíše:

zašifrováno: Hj tcj teyzvvnmz xnhy zm snhy

Samozřejmě pro jakékoliv praktické použití by ovšem výsledný soubor nešifrovaný text ani klíč neobsahoval.

Pro předvedení dešifrování dejme tomu, že jsme dostali zprávu „*Itvo sd jucwu yl yopizd*“, a víme, že klíč je „*pfuk*“. Zadání je pak:

Itvo sd jucwu yl yopizd; pfuk; decrypt

A výsledek?

zašifrovaný text: Itvo sd jucwu yl yopizd klíč: pfuk

dešifrováno: Snad mi tohle da stesti