Рубежный контроль № 2
по теме «Списковые включения, текстовые файлы, работа в консоли»

Весенний семестр

Коновалов А. В.

15 мая 2022 г.

Оглавление

Задачи на списковые включения

Распределение задач по вариантам:

Вариант по списку	Номера задач
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

Задачи должны быть решены без использования циклов `for` и `while`. Следует использовать списковые включения и встроенные функции `map`, `filter`, `max`, `min`, `len`, `sum`.

Запись $x = x_0(h_x)x_n$ в условии задачи означает, что аргументы функции заданы в диапазоне $x_0 \le x \le x_n$ с шагом h_x, т.е. x принимает значения $x_0, x_1 = x_0 + h_x, x_2 = x_0 + 2h_x$ и так далее.

Задача 1. Табулирование функции

1. Значения функций $f_1(x) = 2 + sin\ 3x$ и $f_2(x) = \cos\frac{x}{3}$ определены в точках $x = x_0(h_x)x_n$. Определить, при каком x расстояние между $f_1(x)$ и $f_2(x)$ минимально.
2. Значения функции $f(x) = \frac{1}{2} + \sin\frac{x}{2}$ определены при значениях $x = x_0(h_x)x_n$. Определить, сколько значений функции $f(x)$ при заданных x являются отрицательными и сколько положительными.
3. Значения функций $f_1(x) = 3 + \sin^2\frac{x}{2}$ и $f_2(x) = 1 + \cos^2\frac{x}{3}$ определены в точках $x = x_0(h_x)x_n$. Определить, при каком x расстояние между $f_1(x)$ и $f_2(x)$ максимально.
4. Значения функции $f(x) = 0{,}348 + \cos\frac{x}{4}$ определены при $x = x_0(h_x)x_n$. Определить, сколько значений функции $f(x)$ при заданных x расположены выше прямой $y = 0{,}555$, и сколько ниже той же прямой.
5. Значения функции $f(x) = \sin x$ заданы в точках $x = x_0(h_x)x_n$. Определить количество точек функции $f(x)$, расположенных между прямыми $y = 0{,}5$ и $y = -0{,}5$.
6. Значения функции $f(x) = 2\cos\frac{5x}{4}$ заданы в точках $x = x_0(h_x)x_n$. Определить среднее значение положительных значений функции.
7. Значения функции $f(x) = \sin x + \cos x$ определены при $x = x_0(h_x)x_n$. Определить минимальное значение среди положительных значений функции $f(x)$.
8. Значения функции $f(x) = x^4 + x^2 + x - 3$ заданы в точках $x = x_0(h_x)x_n$. Определить среднее значение для заданных значений функции.
9. Значения функции $f(x) = 3\sin\frac{x}{3} + 2\sin\frac{x}{2}$ заданы в точках $x = x_0(h_x)x_n$. Определить минимальное и максимальное значения функции $f(x)$.
10. Значения функции $f(x) = 3x^4 + 2x^3 - 3x^2 - 3$ заданы в точках $x = x_0(h_x)x_n$. Определить, каких значений функции $f(x)$ больше, положительных или отрицательных.
11. Вычислить сумму $\sum_{n=0}^{10}\frac{x^{4n+1}}{4n+1}$ при $x = 1{,}2$.
12. Вычислить сумму $\sum_{n=1}^{10} n^2 + \sum_{n=1}^{12} n^3$.

Задача 2. Вычисление суммы ряда

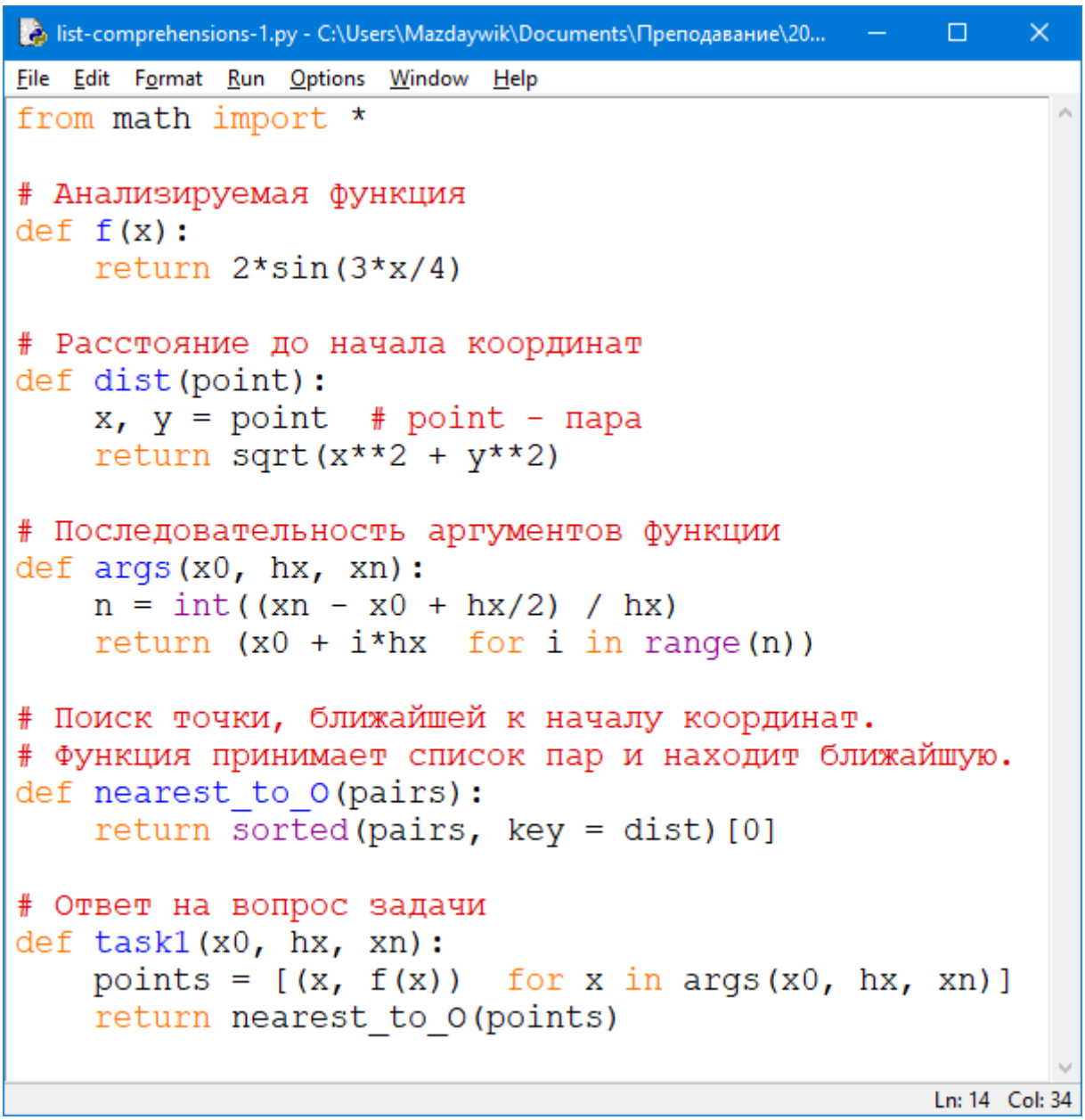
Составить алгоритм и программу вычисления суммы ряда. Справа от знака равенства дано для контроля математическое значение ряда.

1. $1 + 1/2 + 1/4 + 1/8 + \cdots + 1/2^n \approx 2$
2. $1 - 1/2 + 1/4 - 1/8 + \cdots \pm 1/2^n \approx 2/3$
3. $1 - 1/3 + 1/5 - 1/7 + 1/9 - \cdots \pm 1/(2n-1) \approx \pi/4$
4. $\frac{1}{1\cdot2} + \frac{1}{2\cdot3} + \frac{1}{3\cdot4} + \cdots + \frac{1}{n(n+1)} \approx 1$
5. $\frac{1}{1\cdot3} + \frac{1}{3\cdot5} + \frac{1}{5\cdot7} + \cdots + \frac{1}{(2n-1)(2n+1)} \approx 1/2$
6. $\frac{1}{1\cdot3} + \frac{1}{2\cdot4} + \frac{1}{3\cdot5} + \cdots + \frac{1}{(n-1)(n+1)} \approx 3/4$
7. $1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 + \cdots + 1/n^2 \approx \pi^2/6$
8. $1/1^2 + 1/3^2 + 1/5^2 + \cdots + 1/(2n+1)^2 \approx \pi^2/8$
9. $1/1^4 + 1/2^4 + 1/3^4 + \cdots + 1/n^4 \approx \frac{\pi^4}{90}$
10. $1/1^4 - 1/2^4 + 1/3^4 + \cdots \pm 1/n^4 \approx \frac{7\pi^4}{720}$
11. $1/1^4 + 1/3^4 + 1/5^4 + \cdots + 1/(2n+1)^4 \approx \frac{\pi^4}{96}$
12. $1/0! + 1/1! - 1/2! + 1/3! - \ldots \pm 1/n! \approx 1/e$

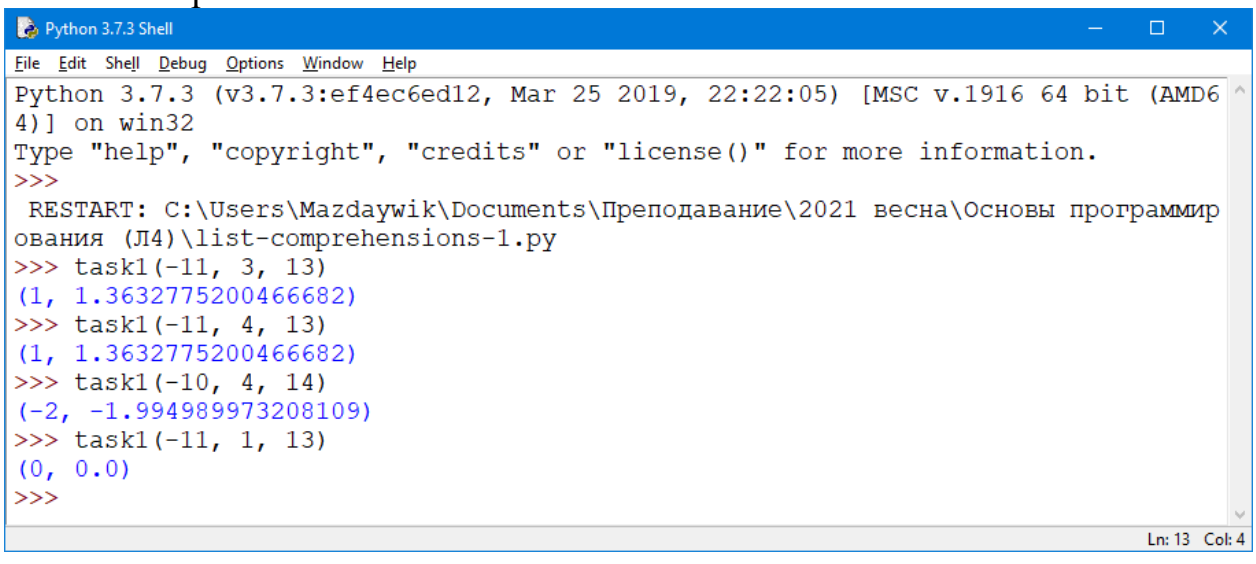
Образец решения задач на списковые включения

Задача 1. Значения функции $f(x) = 2\sin\frac{3x}{4}$ определены при различных значениях аргумента $x = x_0(h_x)x_n$. Определить координаты точки $\big(x, f(x)\big)$, которая ближе всего расположена к началу координат.

Решение. Программа на Python:



Тестирование:



Задача 2. Составить алгоритм и программу вычисления суммы ряда. Справа от знака равенства дано для контроля математическое значение ряда.

$$1 - 1/2 + 1/3 - 1/4 + \ \ldots \ \pm \ 1/n \approx \ln 2.$$

Решение.
Запишем ряд в общем виде:

$$1 - 1/2 + 1/3 - 1/4 + \ \ldots \ \pm \ 1/\mathrm{n} = \sum_{k=0}^{n-1} \frac{(-1)^k}{k+1}.$$

Действительно, первое слагаемое (при $k = 0$) будет иметь вид $1 = \frac{(-1)^0}{0+1}$, последнее слагаемое (при $k = n - 1$) — $\pm 1/n = \frac{(-1)^n}{n}$. Возведение в степень -1 обеспечивает чередование знаков. Нумерация слагаемых с 0 до $n - 1$ позволяет использовать `range(n)` для нумерации слагаемых.

Программа на Python:

list-comprehensions-2.py - D:/Mazdaywik/Documents/MEGA/Преподавание/2021 весна/Основы...

File Edit Format Run Options Window Help

```
def series(n):
    return sum((-1)**k/(k+1)  for k in range(n))
```

Ln: 3 Col: 0

Тестирование:

IDLE Shell 3.9.2

File Edit Shell Debug Options Window Help

```
Python 3.9.2 (tags/v3.9.2:1a79785, Feb 19 2021, 13:44:55) [MSC v.1928 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: D:/Mazdaywik/Documents/MEGA/Преподавание/2021 весна/Основы про
граммирования (Л4)/list-comprehensions-2.py
>>> series(10)
0.6456349206349207
>>> series(100)
0.688172179310195
>>> series(1000)
0.6926474305598223
>>> series(10000)
0.6930971830599583
>>> series(100000)
0.6931421805849816
>>> # точное значение
>>> from math import *
>>> log(2)
0.6931471805599453
>>>
```

Ln: 19 Col: 4

Видно, что при увеличении числа слагаемых сумма ряда приближается к ожидаемому точному значению.

UNIX-утилиты

Командная строка операционной системы UNIX, а также других операционных систем, совместимых с ней (например, Linux или macOS) содержит набор из нескольких десятков служебных программ (утилит), которые предназначены для работы с файлами, сетью, текстовыми данными и самой операционной системы. Многие из этих утилит в качестве входных данных используют либо файлы, указанные в командной строке, либо stdin, а результат своей работы выводят на stdout.

В этой задаче предлагается написать программы на Python, имитирующие работу некоторых UNIX-утилит, предназначенных для работы с текстом.

Написанная программа (кроме задач `tee` (№ 4) и `uniq` (№ 7)) должна получать в командной строке имена файлов, подлежащие обработке и по очереди выполнять с ними требуемую операцию. Если имена файлов отсутствуют, программа должна обрабатывать стандартный ввод.

Распределение задач по вариантам:

Вариант по списку	Номера задач	
1, 22	1	8
2, 23	2	9
3	3	10
4	4	8
5	5	9
6	6	10
7	7	8
8	1	9
9	2	10
10	3	8
11	4	9
12	5	10
13	6	8
14	7	9
15	1	10
16	2	8
17	3	9
18	4	10
19	5	8
20	6	9
21	7	10
22	6	8

1. wc -w (варианты 1, 8, 15, 22)

Утилита wc (word count) может подсчитывать количество слов, строк и символов в указанных файлах, либо в потоке стандартного ввода. Опция командной строки -w (от слова words) предписывает подсчитывать только слова:

```
$ wc -w discount_card.py
     87 discount_card.py
```

Т. е. файл discount_card.py содержит 87 слов. Если указано два или больше файлов, то выводится также и суммарное число слов в нескольких файлах:

```
$ wc -w discount_card.py filescan.py
     87 discount_card.py
     54 filescan.py
    141 total
```

При считывании со стандартного ввода выписывается только общее число слов, имя файла не выписывается:

```
$ wc -w < discount_card.py
     87
```

Требуется написать программу wc-w.py, которая имитирует работу программы wc с опцией -w.

2. wc -l (варианты 2, 9, 16, 23)

Утилита wc (word count) может подсчитывать количество слов, строк и символов в указанных файлах, либо в потоке стандартного ввода. Опция командной строки -l (от слова lines, т. е. строчная буква L, не единица и не i!) предписывает подсчитывать только строки:

```
$ wc -l discount_card.py
     29 discount_card.py
```

Т. е. файл discount_card.py содержит 29 строк. Если указано два или больше файлов, то выводится также и суммарное число строк в нескольких файлах:

```
$ wc -l discount_card.py filescan.py
     29 discount_card.py
     22 filescan.py
     51 total
```

При считывании со стандартного ввода выписывается только общее число строк, имя файла не выписывается:

```
$ wc -l < discount_card.py
     29
```

Требуется написать программу wc-l.py, которая имитирует работу программы wc с опцией -l.

3. tac (варианты 3, 10, 17)

Утилита `tac` распечатывает файл, располагая строки в обратном порядке.

```
$ tac filescan.py
            f.close()
            scan(f, word)
            f = open(fname)
            print("Сканируется файл " + fname)
        for fname in sys.argv[2:]:
    else:
        scan(sys.stdin, word)
        print("Сканируется стандартный ввод")
    if len(sys.argv) == 2:
    word = sys.argv[1]
else:
    sys.stderr.write("Ошибка в командной строке")
if len(sys.argv) < 2:

            print(line[:-1])
        if word in line:
    for line in f:
def scan(f, word):

import sys

# encoding: 866
```

Требуется написать программу `tac.py`, которая имитирует поведение программы `tac`.

4. tee (варианты 4, 11, 18)

Программа `tee` используется для сохранения стандартного вывода в файл с одновременным выводом на экран.

```
$ grep argv filescan.py | tee out.txt
if len(sys.argv) < 2:
    word = sys.argv[1]
    if len(sys.argv) == 2:
        for fname in sys.argv[2:]:
```

При этом файл `out.txt` станет содержать следующий текст:

```
if len(sys.argv) < 2:
    word = sys.argv[1]
    if len(sys.argv) == 2:
        for fname in sys.argv[2:]:
```

То есть, программа `tee` принимает со стандартного ввода некоторый текст, выводит его на стандартный вывод и сохраняет его же в файл, указанный в командной строке.

Следует написать программу `tee.py` с той же функциональностью.

5. sed s/.../.../ (варианты 5, 12, 19)

Утилита `sed` может производить довольно сложные построчные преобразования в текстовых файлах либо в стандартном вводе. Но чаще всего

используют команду `s/исходное-слово/замена/` утилиты `sed`, которая производит замену одного слова на другое (на самом деле и исходное слово, и его замена представляют достаточно сложные шаблоны — регулярные выражения, но здесь мы полагаем, что это просто слова):

```
$ sed s/argv/CHUPAKABRA/ filescan.py
# encoding: 866

import sys

def scan(f, word):
    for line in f:
        if word in line:
            print(line[:-1])

if len(sys.CHUPAKABRA) < 2:
    sys.stderr.write("Ошибка в командной строке")
else:
    word = sys.CHUPAKABRA[1]
    if len(sys.CHUPAKABRA) == 2:
        print("Сканируется стандартный ввод")
        scan(sys.stdin, word)
    else:
        for fname in sys.CHUPAKABRA[2:]:
            print("Сканируется файл " + fname)
            f = open(fname)
            scan(f, word)
            f.close()
```

Требуется написать программу `sed-s.py` которая принимает как минимум два параметра командной строки: исходное слово, его замену и необязательный список файлов, и заменяет в каждой строке указанных файлов (или стандартного ввода, если имена файлов не указаны) первое вхождение слова на его замену.

6. nl (варианты 6, 13, 20)

Программа `nl` выводит содержимое текстовых файлов, указанных в командной строке, или stdin, но с пронумерованными строчками. При этом, если в командной строке указано несколько файлов, нумерация ведётся сквозная.

```
$ nl filescan.py
     1  # encoding: 866
     2
     3  import sys
     4
     5  def scan(f, word):
     6      for line in f:
     7          if word in line:
     8              print(line[:-1])
     9
    10  if len(sys.argv) < 2:
    11      sys.stderr.write("Ошибка в командной строке")
    12  else:
    13      word = sys.argv[1]
```

```
    14      if len(sys.argv) == 2:
    15          print("Сканируется стандартный ввод")
    16          scan(sys.stdin, word)
    17      else:
    18          for fname in sys.argv[2:]:
    19              print("Сканируется файл " + fname)
    20              f = open(fname)
    21              scan(f, word)
    22              f.close()
```

Требуется написать программу, `nl.py`, демонстрирующую заданное поведение. Следует обратить внимание, что оригинальная утилита `nl` выравнивает номера по правой границе и добавляет два пробела между номером и началом строки.

7. uniq (варианты 7, 14, 21)

Утилита `uniq` принимает ноль, одно или два имени файла. Если указано два имени, то первое из них считается входным файлом, второе — выходным. Если указано одно имя файла, результат работы программы пишется на стандартный вывод. Если программа вызвана без аргументов, то входные данные читаются со стандартного ввода.

Утилита `uniq` читает строки из входного файла (либо stdin) и выписывает в выходной файл (или stdout) строки без *соседних* дублирующихся. Пусть в файле `input.txt` находится следующий текст:

```
Повторяется дважды
Повторяется дважды
Написано один раз
Повторяется дважды
Повторяется дважды
Повторяется три раза
Повторяется три раза
Повторяется три раза
Повторяется дважды
Повторяется дважды
Написано один раз
Повторяется три раза
Повторяется три раза
Повторяется три раза
```

Запуск программы `uniq` в консоли тремя способами:

```
$ uniq input.txt uniq.txt
$ uniq input.txt
Повторяется дважды
Написано один раз
Повторяется дважды
Повторяется три раза
Повторяется дважды
Написано один раз
Повторяется три раза
$ cat song.txt | uniq
Повторяется дважды
Написано один раз
Повторяется дважды
Повторяется три раза
```

```
Повторяется дважды
Написано один раз
Повторяется три раза
```

Файл `uniq.txt` будет содержать следующий текст:

```
Повторяется дважды
Написано один раз
Повторяется дважды
Повторяется три раза
Повторяется дважды
Написано один раз
Повторяется три раза
```

Требуется написать программу `uniq.py` с описанной функциональностью.

8. head (варианты 1, 4, 7, 10, 13, 16, 19, 22)

Утилита `head` выводит первые N строк файла или stdin, где значение N указывается в командной строке (число строк начинается с дефиса). Если число строк не указано, выводятся первые 10 строк:

```
$ head filescan.py
# encoding: 866

import sys

def scan(f, word):
    for line in f:
        if word in line:
            print(line[:-1])

if len(sys.argv) < 2:
```

```
$ head -6 filescan.py
# encoding: 866

import sys

def scan(f, word):
    for line in f:
```

Требуется написать программу `head.py`, которая делает тоже самое.

9. tail (варианты 2, 5, 8, 11, 14, 17, 20, 23)

Утилита `tail` выводит последние N строк файла или stdin, где значение N указывается в командной строке (число строк начинается с дефиса). Если число строк не указано, выводятся последние 10 строк:

```
$ tail filescan.py
    word = sys.argv[1]
    if len(sys.argv) == 2:
        print("Сканируется стандартный ввод")
        scan(sys.stdin, word)
    else:
        for fname in sys.argv[2:]:
            print("Сканируется файл " + fname)
            f = open(fname)
```

```
            scan(f, word)
            f.close()
```

```
$ tail -6 filescan.py
    else:
        for fname in sys.argv[2:]:
            print("Сканируется файл " + fname)
            f = open(fname)
            scan(f, word)
            f.close()
```

Требуется написать программу `tail.py`, которая делает тоже самое.

10. expand (варианты 3, 6, 9, 12, 15, 18, 21, 24)

Программа expand предназначена для преобразования символов табуляции в пробелы. Ширина позиций табуляции задаётся опцией `-t`.

Символ табуляции (обозначается как `'\t'`) при отображении на экране заменяется на последовательность из пробелов длиной от 1 до N, где N — размер табуляции. Длина последовательности пробелов определяется таким образом, чтобы символ, следующий за символом табуляции, начинался располагался в позиции k·N – 1, где k — целое число (считается, что первый символ в строке начинается с позиции 1). Например, если величина табуляции равна N = 8, то символы после знака '\t' будут располагаться в позициях 8 + 1 = 9, 2×8 + 1 = 17, 3×8 + 1 = 25 и так далее.

Пример. Пусть дан следующий текст (знаками → обозначены табуляции, точками — пробелы):

```
Страна→Население·(млн·чел.)→Площадь (млн·км²)
Россия→146,5→→→17,1
Китай→1·376,2→→→10,0
США→323,3→→→9,5
```

После преобразования в пробелы (ширина табуляции 8) текст будет выглядеть так:

```
Страна··Население·(млн·чел.)····Площадь (млн·км²)
Россия··146,5···················17,1
Китай···1·376,2·················10,0
США·····323,3···················9,5
```

Пример работы программы expand:

```
$ expand countries.txt
Страна  Население (млн чел.)    Площадь (млн км2)
Россия  146,5                   17,1
Китай   1 376,2                 10,0
США     323,3                   9,5
$ expand -t10 countries.txt
Страна    Население (млн чел.)          Площадь (млн км2)
Россия    146,5                         17,1
Китай     1 376,2                       10,0
США       323,3                         9,5
$ expand -t3 countries.txt
Страна   Население (млн чел.) Площадь (млн км2)
```

```
Россия   146,5       17,1
Китай 1 376,2        10,0
США   323,3       9,5
$ expand -t4 countries.txt
Страна  Население (млн чел.)    Площадь (млн км2)
Россия  146,5           17,1
Китай   1 376,2         10,0
США 323,3           9,5
```

Требуется написать программу `expand.py`, которая выполняет раскрытие табуляций и поддерживает опцию `-t`.

Подсказка: воспользуйтесь методом строки `.expandtabs(n)`.