

Министерство науки и высшего образования РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Братский государственный университет»

А.Н. Ефремова

ПРОГРАММИРОВАНИЕ (II часть)

*Методические указания
по выполнению лабораторных работ*

Братск
Издательство Братского государственного университета
2021

УДК 004

Ефремова А.Н. Программирование (2 часть): методические указания по выполнению лабораторных работ. – Братск: Изд-во БрГУ, 2021. – 104 с.

Даются рекомендации по выполнению лабораторных работ по дисциплине «Программирование» для обучающихся направления 09.03.02 «Информационные системы и технологии» всех форм обучения.

Рецензент **В.А. Мельникова**, канд. техн. наук,
доцент кафедры ИМиФ
(ФГБОУ ВО «БрГУ», г. Братск)

Отпечатано в авторской редакции
в издательстве ФГБОУ ВО «БрГУ»
665709, Братск, ул. Макаренко, 40
Заказ 69

*Электронная версия издания размещена
в локальной сети ФГБОУ ВО «БрГУ»
в разделе «Библиотека»*

© ФГБОУ ВО «БрГУ», 2021
© Ефремова А.Н., 2021

ВВЕДЕНИЕ

Методические указания предназначены для использования обучающимися в процессе выполнения лабораторных работ по дисциплине «Программирование», а также самостоятельной работы.

Процесс изучения дисциплины направлен на формирование компетенции ОПК-6 – способность разрабатывать алгоритмы и программы, пригодные для практического применения в области информационных систем и технологий.

В результате освоения дисциплины обучающийся должен:

знать синтаксис выбранного языка программирования, особенности программирования на этом языке, стандартные библиотеки языка программирования;

уметь составлять алгоритмы, использовать базовые алгоритмические структуры для решения типовых задач; применять выбранные языки программирования для написания программного кода; писать и отлаживать коды на языке программирования, тестировать работоспособность программ;

владеть навыками разработки алгоритмов и программ для решения практических задач в области информационных систем и технологий; навыками выбора, применения методов и алгоритмов и технологии программирования для решения стандартных профессиональных задач; языком программирования высокого уровня, методами отладки и тестирования работоспособности программы.

Лабораторная работа № 7. ПРЕОБРАЗОВАНИЕ СИМВОЛЬНЫХ ВЕЛИЧИН

Цель работы – ознакомиться с основными конструкциями, используемыми для обработки строковой информации; приемами программной реализации на языке программирования Python; произвести отладку и тестирование полученных программ.

Задание:

1. Изучить краткие теоретические сведения.
2. Разработать алгоритмы для решения задач (блок-схемы).
3. Записать алгоритмы на языке программирования Python.
4. Произвести отладку и тестирование программы.
5. Оформить отчет по лабораторной работе.

Отчет о лабораторной работе должен содержать следующие сведения:

- 1) название и цель работы;
- 2) формулировку задачи, схему алгоритма, программный код и результаты решения задачи;
- 3) вывод по работе в целом.

Краткие теоретические сведения

Символьная строка – это последовательность символов.

В языке Python для работы со строками используется специальный тип данных *str* (от английского слова *string*), который позволяет:

- работать с целой символьной строкой как с единым объектом;
- использовать строки переменной длины.

Переменная типа *str* называется строковой или символьной.

Новое значение присваивается строковой переменной с помощью оператора присваивания:

stroka = "Обработка строковой переменной"

Проверить её тип можно следующей командой:

print(type(stroka))

которая выведет

```
<class 'str'>
```

Для ввода значения строковой переменной с клавиатуры используется уже знакомая нам функция **input**:

```
stroka = input()
```

или

```
stroka=input('Введите строку')
```

Встроенная функция **len** определяет длину строки – количество символов в ней. Вот так в переменную *n* записывается длина строки *stroka*:

```
n = len(stroka)
```

Сравнение строк

Строки можно сравнивать между собой так же, как числа. Например, можно проверить равенство двух строк:

```
password = input("Введите пароль:")
if password == "Kykyshka":
    print("Пароль введен верно!")
else:
    print("Пароль введен не верно!")
```

Можно также определить, какая из двух строк больше, какая – меньше. Если строки состоят только из русских или только из латинских букв, то меньше будет та строка, которая идет раньше в алфавитном списке.

Например, слово «*паровоз*» будет «меньше», чем слово «*пароход*»: они отличаются в пятой букве и «*в*» < «*х*». Это можно проверить экспериментально, например, с помощью такой программы:

```
10
11
12 str1 = "паровоз"
13 str2 = "пароход"
14 if str1 < str2:
15     print( str1, "<", str2 )
16 elif str1 == str2:
17     print( str1, "=", str2 )
18 else:
19     print( str1, ">", str2 )
20
```

Сложение и умножение

Оператор «+» используется для «сложения» (объединения, сцепления) строк. Эта операция иногда называется **конкатенация**. Например:

```
9
10
11
12 s1 = "Факультет Энергетики и Автоматики"
13 s2 = "гр.ИСИТ-19"
14 s = s1 + ", " + s2 + "!"
15 print( s )
16
```

В язык Python введена операция умножения строки на число: она заменяет многократное сложение. Например,

```
11
12 s1 = "Сенсация!"
13 s=s1*3
14 print( s )
15
16 |
```

Обращение к символам

В Python каждый символ строки имеет свой номер (индекс), причём нумерация, как и во многих других языках программирования (C, C++, Java), всегда начинается с нуля.

Индекс можно понимать как смещение символа от начала строки. Первый по счёту символ имеет нулевое смещение (находится в самом начале строки), поэтому его индекс – 0:

Индексы	0	1	2	3	4	5	6
S	П	Р	И	В	Е	Т	!

К любому символу можно обратиться по индексу, записав индекс в квадратных скобках после имени строки:

```
11
12 S = "Привет!"
13 print( S[1] ) # p
14 print( S[5] + S[2] + "к" ) # тик
15
```

В языке Python можно указывать отрицательные индексы.

Это значит, что отсчёт ведётся от конца строки, так что символ **S[-1]** – это последний символ строки **S**:

Индексы	-7	-6	-5	-4	-3	-2	-1
S	П	Р	И	В	Е	Т	!

Чтобы рассчитать «обычную» позицию символа в строке, к отрицательному индексу нужно добавить длину строки. Например,

$S[-1] = S[\text{len}(S)-1] = S[6]$

Предыдущую программу можно было переписать, используя отрицательные индексы:

```
10
11
12 S = "Привет!"
13 print( S[-6] )           # p
14 print( S[-2] + S[-5] + "к" ) # тик
15
```

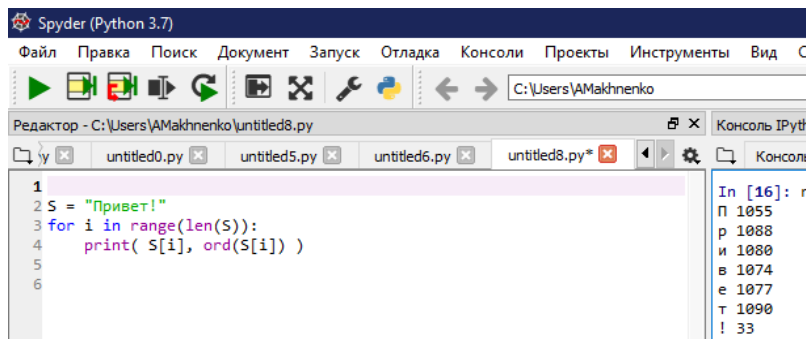
Если указать неправильный индекс, произойдёт ошибка – выход за границы строки, и программа завершится ошибкой. Для строки из семи символов правильные индексы – от «-7» до 6.

Перебор всех символов

Поскольку к символу можно обращаться по индексу, для перебора всех символов можно использовать цикл по переменной, которая будет принимать все возможные значения индексов.

Пусть нужно вывести в столбик коды всех символов строки с именем **S**. Длину строки можно найти с помощью функции **len**, индекс первого символа равен 0, а индекс последнего равен **len(S)-1**. Таким образом, все допустимые индексы – это последовательность, которую строит вызов функции **range(len(S))**.

Перебор можно выполнить так:



```
1
2 S = "Привет!"
3 for i in range(len(S)):
4     print( S[i], ord(S[i]) )
5
6
```

Консоль IPyT

```
In [16]: r
П 1055
р 1088
и 1080
в 1074
е 1077
т 1090
! 33
```

В каждой строке сначала выводится сам символ, а потом – его код, который возвращает встроенная функция **ord**.

В языке Python существует еще один удобный способ перебора всех элементов строки:

```
for c in s:  
print( c, ord(c) )
```

Заголовок такого цикла можно «прочитать» так: «для всех *c*, входящих в состав *s*». Это означает, что все символы строки *s*, с первого до последнего, по очереди оказываются в переменной *c*, и в теле цикла нам остаётся вывести код символа *c*.

В отличие от большинства современных языков программирования, в Python нельзя изменить символьную строку, поскольку строка – это неизменяемый объект. Это значит, что оператор присваивания `s[5]="a"` не сработает – будет выдано сообщение об ошибке.

Тем не менее, можно составить из символов существующей строки новую строку, и внести в неё нужные изменения.

Требуется ввести строку символов с клавиатуры, заменить в ней все буквы «е» на буквы «э» и вывести полученную строку на экран:

```
4 s = input( "Введите строку: " )  
5 s1 = ""  
6 for c in s:  
7     if c == "е": c = "э"  
8     s1 += c  
9 print( s1 )
```

Введите строку: Петя вел беседу
Петя вэл бэсэду
In [20]:

Здесь в цикле

```
for c in S:
```

```
...
```

перебираются все символы, входящие в строку *s*. В теле цикла проверяем значение переменной *c* (это очередной символ исходной строки): если оно совпадает с буквой «е», то заменяем его на букву «э».

```
if c == "е": c = "э"
```

и добавляем в конец новой строки **S1** с помощью оператора сложения:

```
S1 += c
```


Срезы

Для того, чтобы выделить часть строки (подстроку), в языке Python применяется операция получения среза (англ. **slicing**). Например, `s[3:8]` означает «символы строки `s` с 3-го по 7-й» (то есть до 8-го, не включая его).

Следующий фрагмент копирует в строку `s1` символы строки `s` с 3-го по 7-й (всего 5 символов):

```
3
4 s = "0123456789"
5 s1 = s[3:8]
6 print( s1 )
7
```

In [21]
34567

In [22]

Можно использовать и отрицательные индексы – в этом случае отсчёт выполняется с конца строки:

```
2 s = "0123456789"
3 s1 = s[-7:-2] # s1 = "34567"
4 print( s1 )
```

In [22]
34567

Для получения «обычной» позиции символа в строке к отрицательному значению добавляется длина строки. Например, второй индекс «-2» можно заменить на `len(s1)-2`. Это означает, что последние два символа не входят в срез.

Если первый индекс не указан, считается, что он равен нулю (берём начало строки), а если не указан второй индекс, то в срез включаются все символы до конца строки. Например:

```
5 s = "0123456789"
6 sFirst = s[:4] # sFirst = "0123"
7 sLast = s[-4:] # sLast = "6789"
8
9 print( s )
10 print( sFirst )
11 print( sLast )
```

In [24]:
0123456789
0123
6789

In [25]: |

Срезы позволяют легко выполнить *реверс строки* (переставить её символы в обратном порядке):

```
7
8 s = "0123456789"
9 sReversed = s[::-1]
10 print(s)
11 print( sReversed )
12
```

In [27]: run
0123456789
9876543210

In [28]:

Так как начальный и конечный индексы элементов строки не указаны, задействована вся строка. Число «-1» означает шаг изменения индекса и говорит о том, что все символы перебираются в обратном порядке.

Встроенные методы

В Python существует множество встроенных алгоритмов для работы с символьными строками. Многие из них вызываются с помощью точечной записи, они называются **методами обработки строк**.

Например, методы **upper** и **lower** позволяют перевести строку соответственно в верхний и нижний регистр:

```
7
8 s = "aAbBcC"
9 sUp = s.upper() # sUp = "AABBCC"
10 sLow = s.lower() # sLow = "aabbcc"
11 print(sUp)
12 print(sLow)
```

```
In [28]:
AABBCC
aabbcc

In [29]:
```

Слева от точки записывается имя строки (или сама строка в кавычках), к которой нужно применить метод, а справа от точки – название метода. Например, возможна такая запись:

```
7
8 s = "Внимание!".upper() # "ВНИМАНИЕ!"
9 print(s)
10
```

```
In [29]: rl
ВНИМАНИЕ!

In [30]:
```

Здесь метод **upper** применяется к строке «Внимание!».

Методы строк мы уже использовали, когда выводили данные на экран с помощью метода **format**:

```
2
3
4 a = 5
5 b = 4
6 print( "{}+{}={}".format(a,b,a+b) )
7
```

```
In [30]:
5+4=9

In [31]:
```

Ещё один метод, **isdigit**, проверяет, все ли символы строки – цифры, и возвращает логическое значение:

```

4
3 s = "Метка11"
4 print( s.isdigit() ) # False
5
2 s = "123789"
3 print( s.isdigit() ) # True
4

```

In [36]:
False

In [37]:
True

Полезный метод **strip** (по-английски – лишать) удаляет пробелы в начале и в конце строки:

```

2
3
4 sRaw = " Долой дистанционное обучение! "
5 sClear = sRaw.strip() # sClear = "Долой дистанционное обучение!"
6 print(sRaw)
7 print(sClear)

```

In [34]: runfile('C:/Users/AMak
Долой дистанционное обучение!
Долой дистанционное обучение!

In [35]:

Это метод удобно использовать для обработки ввода пользователя, когда пробелы в начале и в конце строки не нужны.

Удаление и вставка

Для удаления части строки нужно составить новую строку, объединив части исходной строки до и после удаляемого участка:

```

1
2 s = "0123456789"
3 s = s[:3] + s[9:]
4 print(s)

```

In [3]
0129

In [3]

Срез **s[:3]** означает «от начала строки до символа **s[3]**, не включая его», а запись **s[9:]** – «все символы, начиная с **s[9]** до конца строки». Таким образом, в переменной **s** остаётся значение «0129».

С помощью срезов и сцепления строк можно также вставить новый фрагмент внутри строки:

```

3
4 s = "0123456789"
5 s = s[:3] + " ,ПРОБЕЛ, " + s[3:]
6
7 print(s)

```

In [40]: runfile('C:/Users
012 ,ПРОБЕЛ, 3456789

In [41]: |

Поиск в символьных строках

В Python существует метод для поиска подстроки (и отдельного символа) в символьной строке, он называется **find** (по-английски – найти). В скобках нужно указать образец для поиска, это может быть один символ или символьная строка:

```
1 s = "Дискретная математика."  
2 n = s.find( "Д" )      # n = 3  
3 if n >= 0:  
4     print( "Номер символа", n )  
5 else:  
6     print( "Символ не найден." )  
7
```

In [42]: runfile(
Номер символа 0

In [43]:

Метод **find** возвращает целое число – индекс символа, с которого начинается образец (буква «с») в строке *s*. Если образец в строке встречается несколько раз, функция находит первый из них. В рассмотренном примере в переменную *n* будет записано число 3.

Аналогичный метод **rfind** (от англ. *reverse find* – искать в обратную сторону) ищет последнее вхождение образца в строку.

Для той же строки *s*, что и в предыдущем примере, метод **rfind** вернёт 12 (индекс последней буквы «с»):

```
1  
2 s = "Дискретная математика."  
3 n = s.rfind( "и" )  
4 print( n )
```

In [
18

Преобразования «строка↔число»

Пусть символьная строка содержит запись числа. С таким значением нельзя выполнять арифметические операции, потому что это символы, а не число.

Для того чтобы с числовыми данными можно было выполнять вычисления, нужно цепочку символов в числовое значение. В языке Python есть встроенные функции для преобразования типов данных, некоторые из них мы уже использовали:

int – переводит строку в целое число;

float – переводит строку в вещественное число;

str – переводит целое или вещественное число в строку.

Приведём пример преобразования строк в числовые значения:

```
s = "123"
n = int( s )
s = "123.456"
x = float( s ) # x = 123.456
```

Если строку не удалось преобразовать в число (например, если в ней содержатся буквы), возникает ошибка и выполнение программы завершается аварийно.

Теперь рассмотрим примеры обратного преобразования:

```
n = 123
s = str( n ) # s = "123"
x = 123.456
s = str( x ) # s = "123.456"
```

Эти операции всегда завершаются успешно, ошибки быть не может.

Функция **str** использует правила форматирования, установленные по умолчанию. При необходимости можно использовать собственное форматирование, например:

```
n = 123
s = "{:5}".format(n) # s = " 123"
```

Здесь значение переменной *n* записано в 5 позициях, то есть в начале строки будут добавлены два пробела.

Для вещественных чисел можно использовать форматы *f* (с фиксированной запятой) и *e* (научный формат, с плавающей запятой):

```
x = 123.456
s = "{:7.2f}".format(x) # s = "123.46"
s = "{:10.2e}".format(x) # s = "1.23e+02"
```

Формат «**7.2f**» обозначает «вывести в 7 позициях с двумя знаками в дробной части», а формат «**10.2e**» – «в научном формате в 10 позициях с двумя знаками в дробной части».

Функция или метод	Назначение
<i>S1 + S2</i>	Конкатенация (сложение строк)
<i>S1 * 3</i>	Повторение строки
<i>S[i]</i>	Обращение по индексу
<i>S[i:j:step]</i>	Извлечение среза
<i>len(S)</i>	Длина строки

Функция или метод	Назначение
S.join (список)	Соединение строк из последовательности str через разделитель, заданный строкой
S1.count (S[, i, j])	количество вхождений подстроки s в строку s1. Результатом является число. Можно указать позицию начала поиска i и окончания поиска j
S.find (str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.index (str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex (str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace (шаблон, замена)	Замена шаблона
S.split (символ)	Разбиение строки по разделителю
S.upper ()	Преобразование строки к верхнему регистру
S.lower ()	Преобразование строки к нижнему регистру
S.rfind (sub[, start[, end]])	Возвращает индекс первого найденного вхождения при поиске справа
S.isalpha ()	Определяет: состоит ли строка целиком из буквенных символов
S.isdigit ()	Определяет: состоит ли строка целиком из цифр
S.rjust (width[, fillchar = ' '])	Расширяет строку, добавляя символы слева
S.ljust (width[, fillchar = ' '])	Расширяет строку, добавляя символы справа
S.strip ()	Удаляет пробелы и переносы строк справа и слева
S.rstrip ()	Удаляет пробелы и переносы строк справа
S.lstrip ()	Удаляет пробелы и переносы строк слева

Основные функции для работы со строками:

str([<Объект>]) – преобразует любой объект в строку. Если параметр не указан, то возвращается пустая строка;

len(<Строка>) – возвращает количество символов в строке.

Основные методы для работы со строками:

strip([<Символы>]) – удаляет указанные в параметре символы в начале и в конце строки. Если параметр не задан, удаляются пробельные символы: пробел, символ перевода строки («\n»), символ возврата каретки («\r»), символы горизонтальной («\t») и вертикальной («\v») табуляции;

split([<Разделитель>[, <Лимит>]]) – разделяет строку на подстроки по указанному разделителю и добавляет эти подстроки в список, который возвращается в качестве результата. Если первый параметр не указан или имеет значение None, то в качестве разделителя используется символ пробела. Во втором параметре можно задать количество подстрок в результирующем списке – если он не указан или равен -1, в список попадут все подстроки. Если подстрок больше указанного количества, то список будет содержать еще один элемент – с остатком строки;

rsplit([<Разделитель>[, <Лимит>]]) – аналогичен методу **split**(), но поиск символа-разделителя производится не слева направо, а, наоборот, справа налево;

splitlines([True]) – разделяет строку на подстроки по символу перевода строки («\n») и добавляет их в список. Символы новой строки включаются в результат, только если необязательный параметр имеет значение True. Если разделитель не найден в строке, то список будет содержать только один элемент;

partition(<Разделитель>) – находит первое вхождение символа разделителя в строку и возвращает кортеж из трех элементов: первый элемент будет содержать фрагмент, расположенный перед разделителем, второй элемент – сам разделитель, а третий элемент – фрагмент, расположенный после разделителя. Поиск производится слева направо. Если символ-разделитель не найден, то первый элемент кортежа будет содержать всю строку, а остальные элементы останутся пустыми;

join() – преобразует последовательность в строку. Элементы добавляются через указанный разделитель. Формат метода:

<Строка> = <Разделитель>.join(<Последовательность>)

find() – ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то возвращается значение -1. Метод зависит от регистра символов. Формат метода:

<Строка>.find (<Подстрока> [, <Начало> [, <Конец>]])

Если начальная позиция не указана, то поиск будет осуществляться с начала строки. Если параметры Начало и Конец указаны, то производится операция извлечения среза и поиск подстроки будет выполняться в этом фрагменте;

index() – метод аналогичен методу **find()**, но если подстрока в строку не входит, то возбуждается исключение **ValueError**. Формат метода:

`<Строка>.index(<Подстрока>[, <Начало>[, <Конец>]])`

count() – возвращает число вхождений подстроки в строку. Если подстрока в строку не входит, то возвращается значение 0. Метод зависит от регистра символов. Формат метода:

`<Строка>.count(<Подстрока>[, <Начало>[, <Конец>]])`

replace() – производит замену всех вхождений заданной подстроки в строке на другую подстроку и возвращает результат в виде новой строки. Метод зависит от регистра символов. Формат метода:

`<Строка>.replace(<Подстрока для замены>, <Новая подстрока> [, <Максимальное количество замен>])`

Если количество замен не указано, будет выполнена замена всех найденных подстрок.

Пример. Проверить, будет ли строка читаться одинаково справа налево и слева направо (т. е. является ли она палиндромом).

Решение. Сначала введём строку командой: `s=input('Введите строку ')`.

Затем определим логическую переменную `flag` и присвоим ей значение 1: `flag=1`.

Для начала в введённой строке нужно удалить пробелы. Для этого воспользуемся циклической конструкцией `for`, которая выполнится столько раз, какую имеет длину строка. Длину строки определим функцией `len(s)`.

В теле цикла будем проверять следующее условие: `s[i]!=' '`. Данное логическое выражение будет истинно в том случае, если i -й элемент строки не будет равен пробелу, тогда выполнится команда следующая после двоеточия: `string+=s[i]`.

К строке `string`, которая была объявлена в начале программы, будет добавляться посимвольно строка `s`, но уже без пробелов.

Для проверки строки на "палиндром" воспользуемся циклической конструкцией `for`.

Длина половины строки находится делением нацело на 2. Если количество символов нечетно, то стоящий в середине не учитывается, т. к. его сравниваемая пара – он сам.

Количество повторов цикла равно длине половины строки. Длину строки определим функцией `len(s)`, где аргумент введенная нами строка `s`. Зная длину строки, можно вычислить количество повторов цикла. Для этого целочисленно разделим длину строки на 2: `len(s)//2`.

Для задания диапазона для цикла используем функцию `range()`, в которой аргументом будет являться половина длины строки: `range(len(s)//2)`.

`for i in range(len(s)//2):`

Если символ с индексом `i` не равен «симметричному» символу с конца строки (который находится путем индексации с конца) `if s[i] != s[-1-i]`, то переменной `flag` присваивается значение 0 и происходит выход из цикла командой `break`.

Далее, при помощи условной конструкции `if-else` в зависимости от значения `flag` либо – 0, либо -1 выводится сообщение, что строка палиндром, либо нет.

Программа на языке программирования Python:

```
1 s = input("Введите строку\n")
2 flag=1
3 string=""
4 for i in range(len(s)):
5     if s[i]!=' ':
6         string+=s[i]
7 print(string)
8 for i in range(len(s)//2):
9     if string[i]!=string[-i-1]:
10        flag=0
11        break
12 if flag:print('Палиндром')
13 else:print('Не палиндром')
```

```
а роза упала на лапу азора
арозаупаланапуазора
Палиндром
```

```
In [53]: |
```

Задачи для самостоятельной работы

1. Дана строка символов. Сформировать новую строку, состоящую из символов с номерами три, шесть, девять и т. д. данной строки.

2. Даны две строки. Вывести большую по длине строку столько раз, на сколько символов отличаются строки.

3. Дана строка символов. Определите, какой из символов «s» или «q» встречается в ней раньше. Если какой-то из символов в строке отсутствует, то сообщить об этом.

4. Написать программу, которая считает число слов в предложении.

5. Написать программу, которая считает число слов в предложении, начинающихся на заданную букву.

6. Написать программу, которая проверяет, можно ли из букв, входящих в слово А, составить слово В.

7. Дана строка, в которой нет начальных и конечных пробелов. Необходимо изменить ее так, чтобы длина строки стала равна заданной длине, большей, чем текущая длина строки. Это следует сделать путем вставки между словами дополнительных пробелов. Количество пробелов между отдельными словами не должно отличаться более, чем на один пробел (то есть пробелы добавляются равномерно).

8. Написать программу, которая удаляет в данном тексте все лишние пробелы.

9. Составить список слов, которые входят в данное предложение.

10. Подсчитать, из скольких разных слов состоит данное предложение.

11. Подсчитать число различных гласных (согласных), входящих в данное слово (предложение).

12. Упорядочить данный набор слов по алфавиту.

13. Проверьте правильность расстановки скобок в выражении. Алгоритм может быть, например, таким: при подсчете скобок слева направо число открывающих скобок не должно быть меньше числа закрывающих, причем при завершении подсчета эти числа должны совпадать.

14. Из данного предложения выбрать слова, имеющие заданное число букв.

15. Составить программу, подсчитывающую число предложений в тексте (предложение оканчивается символами «.», «?», «!»).

16. Составить программу, вычеркивающую из предложения данное слово.

17. Из заданного набора слов выбрать все слова, имеющие рифмы (рифма определяется по принципу, придуманному Незнай-

кой: два слова рифмуются, если последние слоги у них совпадают, например, «палка – селедка»).

18. Заменить в данном тексте все малые латинские буквы на заглавные (то же самое для русских букв).

19. Перепечатать заданный текст, удалив из него знаки «+», непосредственно за которыми идет цифра.

20. Перепечатать заданный текст, удалив из него все буквы «b», непосредственно перед которыми находится буква «c».

21. Написать программу, генерирующую строки длины 10, причем первые 4 символа – цифры, следующие два символа – различные буквы, следующие 4 символа – нули или единицы, причем одна единица точно присутствует.

22. Дана непустая последовательность непустых слов из латинских букв; соседние слова отделяются друг от друга запятой, за последним словом – точка. Определить количество слов, которые содержат ровно три буквы «e».

23. Возьмем произвольное слово и проделаем с ним следующую операцию: поменяем местами его первую согласную букву с последней согласной буквой, вторую согласную букву с предпоследней согласной буквой и т. д. Если после этой операции мы вновь получим исходное слово, то будем называть такое слово негласным палиндромом. Например, слова «sos», «rare», «rotor», «gong», «kagaake» являются негласными палиндромами. Определить, является ли заданное слово негласным палиндромом.

24. Дан текст. Если первый символ текста не является малой латинской буквой, то оставить его без изменения. Если же это малая латинская буква, но за начальной группой малых латинских букв не следует цифра, то также оставить текст без изменения. Иначе каждую цифру, принадлежащую группе цифр, следующей за начальной группой малых латинских букв, заменить символом «*».

25. Дан текст, каждый символ которого может быть малой буквой, цифрой или одним из знаков «+», «-», «*». Группой букв будем называть такую совокупность последовательно расположенных букв, которой непосредственно не предшествует и за которой непосредственно не следует буква. Аналогично определим группу цифр и группу знаков. Если в данном тексте имеется не менее двух групп букв, то каждый знак «+», встречающийся между двумя группами букв, заменить цифрой 1, знак «-» заменить цифрой 2, а знак «*» – цифрой 3. Иначе оставить текст без изменений.

26. Дана строка, состоящая из слов, разделенных символами, которые перечислены во второй строке. Получить все слова.

27. С клавиатуры вводятся две строки. Найти количество вхождений одной (являющейся подстрокой) в другую.

28. Во введенной строке удалить пробелы между первым и вторым вопросительным знаком.

29. Дана строка слов, разделенных пробелами. Между словами может быть несколько пробелов, в начале и конце строки также могут быть пробелы. Требуется преобразовать строку так, чтобы в ее начале и конце пробелов не было, а слова были разделены одиночным символом «*» (звездочка).

30. Найти слово, стоящее в тексте под определенным номером, и вывести его первую букву.

31. Дана строка символов. Группы символов, разделенные одним или несколькими пробелами и не содержащие пробелов внутри себя, будем называть словами. В самом длинном слове заменить все буквы «а» на «b».

32. Дана строка, состоящая из слов, разделенных пробелами и знаками препинания. Определить длину самого короткого слова.

33. Дана строка слов. Заменить последние три символа слов, имеющих выбранную длину на символ «\$».

34. В произвольной строке удалить последнее слово.

35. Добавить в строку пробелы после знаков препинания, если они там отсутствуют.

36. Найти в строке определенную последовательность символов (подстроку) и заменить ее другой.

37. Дана строка из символов латинского алфавита. Вставьте пробел перед каждой заглавной буквой. Перед первой буквой пробел добавлять не надо.

Пример. Исходная строка:

AtTimesYouMayWantToReadDataFromTheKeyBoard

Полученная строка: At Times You May Want To Read Data From The Key Board

38. Дана строка. Подсчитать общее количество содержащихся в ней строчных латинских и русских букв.

39. Дана строка. Если она представляет собой запись целого числа, то вывести 1, если вещественного (с дробной частью) – вывести 2; если строку нельзя преобразовать в число, то вывести 0.

40. Дана строка, изображающая целое положительное число. Вывести знакочередующуюся сумму цифр этого числа. Пример: для числа 294 сумма равна -3 ($2-9+4=-3$).

41. Дана строка, содержащая натуральное число. Получить строку, содержащую соответствующее число в двоичной системе счисления.

42. Дана строка, содержащая натуральное число, представленное в двоичной системе счисления. Получить строку, содержащую соответствующее число в десятичной системе счисления.

43. Даны целые положительные числа N1 и N2 и строки S1 и S2. Получить из этих строк новую строку, содержащую первые N1 символов строки S1 и последние N2 символов строки S2 (в указанном порядке).

44. Дан символ С и строки S, S0. Перед каждым вхождением символа С в строку S вставить строку S0.

Вариант	№ задания
1	2, 42, 37
2	4, 44, 39
3	6, 1, 41
4	8, 3, 43
5	10, 5, 17
6	12, 7, 22
7	14, 9, 19
8	16, 11, 27
9	18, 13, 24
10	20, 15, 34
11	22, 17, 25
12	24, 19, 6
13	26, 21, 3
14	28, 23, 12
15	30, 25, 1
16	32, 27, 20
17	34, 29, 8
18	36, 31, 14
19	38, 33, 30
20	40, 35, 32

Лабораторная работа № 8. СТРУКТУРЫ ДАННЫХ. СПИСКИ. КОРТЕЖИ

Цель работы – познакомиться с основными конструкциями, используемыми для работы со списками, кортежами; приемами программной реализации на языке программирования Python; произвести отладку и тестирование полученных программ.

Задание:

1. Изучить краткие теоретические сведения.
2. Разработать алгоритмы для решения задач (блок-схемы).
3. Записать алгоритмы на языке программирования Python.
4. Произвести отладку и тестирование программы.
5. Оформить отчет по лабораторной работе.

Отчет о лабораторной работе должен содержать следующие сведения:

- 1) название и цель работы;
- 2) формулировку задачи, схему алгоритма, программный код и результаты решения задачи;
- 3) вывод по работе в целом.

Краткие теоретические сведения

Списки, кортежи, множества и диапазоны – это нумерованные наборы объектов. Каждый элемент набора содержит лишь *ссылку на объект* – по этой причине они могут содержать объекты произвольного типа данных и иметь неограниченную степень вложенности. Позиция элемента в наборе задается индексом. Нумерация элементов начинается с **0**, а не с 1.

Списки и кортежи являются просто упорядоченными последовательностями элементов. Как и все последовательности, они поддерживают обращение к элементу по индексу, получение среза, конкатенацию (оператор «+»), повторение (оператор «*»), проверку на вхождение (оператор **in**) и не вхождение (оператор **not in**).

Списки относятся к изменяемым типам данных. Это означает, что мы можем не только получить элемент по индексу, но и изменить его.

Кортежи относятся к неизменяемым типам данных. Иными словами, можно получить элемент по индексу, но изменить его нельзя, т. е. кортеж – это **неизменяемый список**.

Для добавления и удаления элементов списка используются следующие методы:

append(<Объект>) – добавляет один объект в конец списка. Метод изменяет текущий список и ничего не возвращает;

extend(<Последовательность>) – добавляет элементы последовательности в конец списка. Метод изменяет текущий список и ничего не возвращает;

insert (<Индекс>, <Объект>) – добавляет один объект в указанную позицию. Остальные элементы смещаются. Метод изменяет текущий список и ничего не возвращает;

pop([<Индекс>]) – удаляет элемент, расположенный по указанному индексу, и возвращает его. Если индекс не указан, то удаляет и возвращает последний элемент списка. Если элемента с указанным индексом нет, или список пустой, возбуждается исключение `IndexError`;

remove(<Значение>) – удаляет первый элемент, содержащий указанное значение. Если элемент не найден, возбуждается исключение `ValueError`. Метод изменяет текущий список и ничего не возвращает;

clear() – удаляет все элементы списка, очищая его. Никакого результата при этом не возвращается.

Укажем еще ряд функций и методов, которые будут полезны при работе со списками.

Чтобы узнать индекс элемента внутри списка, следует воспользоваться методом **index**(). Формат метода:

index(<Значение>[, <Начало>[, <Конец>]])

Метод возвращает индекс элемента, имеющего указанное значение. Если значение не входит в список, то возбуждается исключение `ValueError`. Если второй и третий параметры не указаны, то поиск будет производиться с начала и до конца списка.

Узнать общее количество элементов с указанным значением позволяет метод **count**(<Значение>). Если элемент не входит в список, то возвращается значение **0**.

С помощью функций **max**() и **min**() можно узнать максимальное и минимальное значение списка, соответственно.

Метод **reverse()** изменяет порядок следования элементов списка на противоположный. Метод изменяет текущий список и ничего не возвращает.

Отсортировать список позволяет метод **sort()**. Он имеет следующий формат:

sort([key=None][, reverse=False])

Все параметры не являются обязательными. Метод изменяет текущий список и ничего не возвращает.

Чтобы отсортировать список по убыванию, следует в параметре **reverse** указать значение **True**.

В параметре **key** можно указать функцию, выполняющую какое-либо действие над каждым элементом списка. В качестве единственного параметра она должна принимать значение очередного элемента списка, а в качестве результата – возвращать результат действий над ним. Этот результат будет участвовать в процессе сортировки, но значения самих элементов списка не изменятся.

Пример. Список фактически представляет собой индексированную последовательность ссылок на объекты. Объект может быть числом, строкой, списком, словарём, выражением, функцией и т. д. Поэтому в одном списке могут присутствовать объекты разных типов:

A = [17, 'one', [3/5, sqrt(5.29)], -12.3]

Результат: [17, 'one', [0.6, 2.3], -12.3]

Индексация объектов списка начинается с нуля. Доступ к элементу списка:

имя_списка[индекс_элемента]

Для данного примера: значение A[0] равно 17; значение A[1] равно 'one', A[2] есть список [0.6, 2.3].

В том случае, если элемент списка в свою очередь является списком, то элемент внутреннего списка определяется вторым индексом. Так, значение A[2][0] равно 0.6, значение A[2][1] равно 2.3. Возможная глубина вложенности списков намного больше, чем может понадобиться программисту при написании программ.

Если элемент списка является строкой, то возможно обращение к конкретному символу строки.

Так, для примера, рассмотренного выше, значение `A[1][0]` равно 'o', значение `A[1][1]` равно 'n', значение `A[1][2]` равно 'e'. Напомним, что строка является **неизменяемым объектом**, поэтому мы **не можем** изменить отдельный символ строки!

Создание и копирование списков

Покажем на примерах различные способы создания списков.

а) создание пустого списка:

`A = []`

`B = list()`

б) создание списка из n одинаковых элементов:

`A = n*[None]` – список из n значений `None` – «заготовка» будущего списка;

`B = n*[0]` – список из n нулевых значений;

`C = n*[""]` – список из n пустых строк;

в) создание списка из последовательности значений:

`A = [1, 2, 3, 4, 5]` – список целых чисел;

`B = ['строка1', 'строка2', 'строка3']` – список строк;

г) создание списка из существующего объекта X :

`A = list(X)` – поверхностное копирование списка X в новый список (но не `A = X` ! – смотри замечание ниже);

`B = list(range(6))` – создание списка с помощью итератора `range`. Результат: `B = [0, 1, 2, 3, 4, 5]`;

`C = list('string')` – новый список содержит символы строки. Результат: `C = ['s', 't', 'r', 'i', 'n', 'g']`;

`D = list(f(x) for x in X)` или `D = [f(x) for x in X]` – с помощью перечисления элементов итерируемого объекта X ;

`E = X[n:m:k]` – выборка значений из списка X в новый список, что эквивалентно варианту `E = [X[i] for i in range(n, m, k)]`.

Замечание. При выполнении присваивания типа `A = X`, где X – список, в `A` запишется ссылка на объект X . Таким образом, на одно и то же место в памяти будут указывать две переменные. Изменить объект теперь можно через любую переменную.

Пример

```
12 X = [4, 7, -2]
13 A = X
14 A[0] = 12
15 print(X)
```

```
[12, 7, -2]
```

```
In [9]:
```

Следует подробнее остановиться на понятии «поверхностное копирование» списка. В том случае, когда элементами списка являются неизменяемые объекты – числа и строки, – поверхностное копирование создаёт совершенно самостоятельный новый список с копиями значений объектов-элементов списка. Однако если список содержит подписки, то элементами списка являются ссылки – указатели на эти подписки, которые (указатели) и копируются в новый список. Следовательно, если изменения происходят *внутри* подписки, то они отразятся и в копии списка!

Пример

Замена подписки новым объектом

```
5
6 A=[[1,2,3]]; B=list(A)
7 A[0]=[4,5,6]
8 print('A=',A)
9 print('B=',B)
```

In [3]: runfile(
A= [[4, 5, 6]]
B= [[1, 2, 3]]

Изменения внутри подписки

```
6 A=[[1,2,3]]; B=list(A)
7 A[0][0]=0
8 print('A=',A)
9 print('B=',B)
```

In [4]: runfile(''
A= [[0, 2, 3]]
B= [[0, 2, 3]]

Операция * (создание списка из одинаковых значений) также использует поверхностное копирование, поэтому её не рекомендуется использовать для создания списка подписков.

В отличие от поверхностного «глубокое» копирование создаёт полностью независимый список, какой бы глубины вложенности подписков не содержал список. Для глубокого копирования используется метод **deepcopy()** из модуля **copy**.

```
6 from copy import *
7 A=[[1,2,3]]; B=deepcopy(A)
8 A[0][0]=0
9 print('A=',A, 'B=',B)
```

In [5]: runfile('C:/Users/AMa
A= [[0, 2, 3]] B= [[1, 2, 3]]
In [6]: |

Пример

Список.append(объект) – добавляет элемент в конец списка

```
6 A=['6', 'o', 'p']
7 B=['o', 'd', 'a']
8 A.append(B)
9 print('A =', A)
```

A = ['6', 'o', 'p', ['o', 'd', 'a']]
In [7]:

`Список.extend(список)` – «расширяет» список добавлением к нему списка-аргумента

```
9 A=['6','o','p']
10 B=['o','d','a']
11 A.extend(B)
12 print('A =', A)
```

```
A = ['6', 'o', 'p', 'o', 'd', 'a']
In [8]:
```

`Список.insert(индекс, объект)` – добавляет объект в позицию списка, заданную индексом;

`Список.pop(индекс)` – возвращает значение элемента с данным индексом, а затем удаляет его из списка; при отсутствии индекса метод `pop()` применяется к последнему элементу списка.

Пример. Дан список целых чисел. Преобразовать его таким образом, чтобы сначала шли отрицательные значения в порядке их появления в исходном списке, а затем неотрицательные значения также в порядке их появления в исходном списке.

```
24
25 A=[3,-5,8,-7,6,-4]
26 n=len(A)
27 p=0
28 for i in range(n):
29     if A[i]<0:
30         A.insert(p, A.pop(i))
31         p+=1
32 print(A)
```

```
[-5, -7, -4, 3, 8, 6]
In [13]:
```

Преобразование списка одновременно с просмотром его элементов следует проводить с аккуратностью. Пусть, например, необходимо удалить все элементы списка, пользуясь методом `remove()`.

```
5 A=[1,2,3,4,5,6]
6 for a in A:
7     A.remove(a)
8 print('A =', A)
```

```
A = [2, 4, 6]
In [14]:
```

Из результата видно, что в списке останется каждый второй элемент `A = [2, 4, 6]`!

Чтобы получить правильный результат следует обновлять список, создавая его новую копию при каждой итерации цикла:

```
6 A=[1,2,3,4,5,6]
7 for a in list(A):
8     A.remove(a)
9     print('A =', A)
```

```
A = [2, 3, 4, 5, 6]
A = [3, 4, 5, 6]
A = [4, 5, 6]
A = [5, 6]
A = [6]
A = []
```

Теперь каждый раз будет удаляться первый элемент списка.

Список.sort() – сортирует элементы списка «на месте»

Метод `sort()`, как и любой метод, преобразующий исходный список «на месте», возвращает значение `None`, поэтому фактически вызов метода `sort()` есть вызов процедуры. В отличие от метода `sort()` вызов внешней функции `sorted(список)` возвращает отсортированный список, но исходный список оставляет без изменения.

Метод sort()

```
6
7 A=[17,-9, 2]
8 print(A.sort())
9 print(A)
```

```
None
[-9, 2, 17]

In [17]:
```

Функция sorted()

```
11 A=[17,-9, 2]
12 print(sorted(A))
13 print(A)
```

```
[-9, 2, 17]
[17, -9, 2]

In [18]:
```

По умолчанию метод **sort()** и функция **sorted()** сортируют элементы списка по возрастанию. Если необходимо сортировать его по убыванию, то следует указать значение необязательного аргумента **reverse=True**:

```
7 A=[17,-9, 2]
8 print(sorted(A, reverse=True))
```

```
[17, 2, -9]

In [19]:
```

Сортировать можно списки любых объектов, для которых определена операция сравнения. К таким объектам относятся, естественно, числа и строки (но по отдельности!) При попытке сортировки списка, содержащего одновременно и числа, и строки, получим сообщение об ошибке **TypeError: unorderable types**.

Сортировать можно и более сложные объекты, например списки списков.

```
8
9 A=[[6,2,5], [1,14], [8,3], [1,6]]
10 A.sort()
11 print(A)
```

```
In [19]: runfile('C:/Users/AMakhnenko/un',
[[1, 0, 3], [1, 14], [6, 2, 5], [8, 3]]

In [20]:
```

Из примера видно, что подписки отсортированы по возрастанию элементов с индексом 0, а при совпадении их значений сравниваются элементы с индексом 1 и т.д. Список подписков можно отсортировать по любому другому индексу, указав при вызове функции `sort()` необязательный аргумент `key=ключ`, где `ключ` – это имя функции, возвращающей значение элемента подписка с нужным индексом (соответствующий элемент должен присутствовать в каждом подписке!).

Список `reverse()` – реверсирует исходный список «на месте», т.е. переставляет элементы списка в обратном порядке.

Кроме метода `reverse()` существует и функция-итератор `reversed(список)`, которая последовательно возвращает элементы списка от конца к началу.

<pre>3 A=[1,2,3,4,5] 4 A.reverse() 5 print(A)</pre>	<pre>In [44]: print(A) [5, 4, 3, 2, 1] In [33]:</pre>
<pre>3 4 A=[1,2,3,4,5] 5 for a in reversed(A): 6 print(a)</pre>	<pre>In [20] 5 4 3 2 1</pre>

Список также можно использовать для моделирования разнообразных структур данных, в частности, массивов, стеков, очередей и деревьев.

Для включения элемента в стек используем метод `append()`, для извлечения элемента из стека – метод `pop()`. При попытке извлечь элемент из пустого стека, возникает **ошибка `IndexError: pop from empty list`**, поэтому перед извлечением элемента следует проверять стек на не-пустоту (или обрабатывать возникающее при этом исключение).

Задача. Строка содержит правильную последовательность открывающих и закрывающих скобок. Пронумеруем их от 0 – отдельно открывающие, отдельно закрывающие скобки. Требуется для каждой открывающей скобки установить номер соответствующей ей закрывающей скобки.

Решение. Будем просматривать строку символов слева направо. Если текущий символ есть открывающая скобка, то её номер помещаем в стек. Если текущий символ есть закрывающая скобка,

то из стека извлекаем номер «парной» открывающей скобки. Этот номер удобно использовать в качестве индекса при записи результата (номера закрывающей скобки) в массив.

```

2
3 S='(((())())())'
4 p=0 # номер открывающей скоб
5 m=0 # номер закрывающей скоб
6 stack=[]
7 n=len(S)//2
8 A=n*[None]
9 for ch in S:
10     if ch=='(':
11         stack.append(p)
12         p+=1
13     elif ch==')':
14         top=stack.pop()
15         A[top]=m
16         m+=1
17 print(A)

```

Результат:

```

| [5, 4, 0, 2, 1, 3]
  0 1 2 3 4 5

```

внизу добавлены номера соответствующих открывающих скобок.

Задача. Вычислить значение F_n , если $F_0=0$, $F_1=1$, $F_k=F_{k-1}+F_{k-2}$ (такая последовательность называется числами Фибоначчи).

Решение.

```

8 n=int(input('n='))
9 F=[0,1]
10 for k in range(1,n):
11     F.append(sum(F))
12     F.pop(0)
13 print('Fn =', F[1])

```

```

n=66
Fn = 27777890035288

In [28]:

```

Кортеж можно рассматривать как список с неизменяемыми элементами. Синтаксически кортеж отличается от списка использованием круглых скобок вместо квадратных. Однако для распознавания списка интерпретатором скобки не обязательны, достаточно запятых. Отсюда вытекает некоторая странность синтаксиса для кортежа из одного элемента.

$A = 1$, – кортеж из одного элемента, так как при отсутствии запятой A будет иметь тип **int**. Для повышения наглядности рекомендуется всё же при создании кортежа обязательно использовать круглые скобки.

Доступ к элементам кортежа аналогичен доступу к элементам списка:

```

8 A=(0,1.7,[2,3],'end')
9 print(A[2][0], A[3])

```

```

2 end

```

Но, как уже было отмечено, изменить значение элемента кортежа невозможно.

Так, если в предыдущем примере попытаться выполнить оператор присваивания `A[2] = 7`, то появится сообщение об ошибке: **TypeError: 'tuple' object does not support item assignment.**

Однако если элемент кортежа является изменяемым объектом, то можно изменить «содержание» этого объекта. Так, присваивание `A[2][0] = 7` корректно (список является изменяемым объектом), а присваивание `A[3][0]='R'` – нет (строка является неизменяемым объектом).

Для просмотра элементов кортежа можно использовать цикл «для каждого».

Пример:

```
8 A=(0,1.7,[2,3], 'end')
9 for a in A:
10     print(a, '\t', type(a))
```

```
In [31]: runfile('C:/Users
0         <class 'int'>
1.7       <class 'float'>
[2, 3]    <class 'list'>
end       <class 'str'>
```

Для создания кортежа можно использовать те же способы, что и для создания списка с заменой ключевого слова **list** на **tuple**, например, `A=tuple(объект)` – создание кортежа на основе существующего объекта.

Из всех методов списка к кортежам применимы только **count()** и **index()**.

Пример. При попытке выполнить программу, получаем ошибку:

```
8 A=(1,2,3)
9 A.append(4)
10 print(A)
```

```
AttributeError: 'tuple' object has no attribute 'append'
```

Для «наращивания» кортежа можно использовать операцию конкатенации. В этом случае из двух кортежей создаётся новый, а исходные кортежи не изменяются. Пример:

```
11 A = (1, 2, 3)
12 B = A + (4,)
13 print(B)
```

```
(1, 2, 3, 4)
In [34]:
```

Кортежи чаще всего являются вспомогательной структурой данных. В частности, при каждом обращении к функции `zip(A, B, C, ...)` возвращается кортеж вида `(a, b, c, ...)`:

```
10 A=[1,2,3]
11 B=[7,8,9]
12 for x in zip(A,B):
13     print(x, end=' ')
```

```
In [34]: runfile('C:/Us
(1, 7) (2, 8) (3, 9)
In [35]:
```

Задачи для самостоятельного решения

1. Найдите модуль разности первого и последнего элементов числового списка.

2. Определите индекс элемента числового списка, значение которого равно сумме первого и последнего элемента того же списка.

3. Определите, равен ли «центральный» элемент числового списка произведению крайних элементов.

4. Определите, является ли произведение элементов числового списка факториалом числа, равного длине списка.

5. Определите, все ли элементы числового списка являются трёхзначными числами.

6. Определите, все ли элементы числового списка являются чётными числами.

7. По заданному списку целых чисел создайте список, состоящий из их остатков от деления на 2.

8. Определите, встречается ли указанный своим индексом элемент первого заданного списка во втором заданном списке.

9. Составьте программу, которая по двум спискам, имеющим одинаковую длину, возвращает:

1) первый список, если сумма элементов первого списка совпадает с суммой элементов второго списка;

2) сумму элементов второго списка в противном случае.

10. Напишите программу, определяющую в списке, содержащем списки целых чисел, количество таких списков, содержащих заданное число.

11. (Покер) Задан список из пяти чисел. Если одинаковы 5 чисел, то напечатайте число 1, иначе если одинаковы 4 числа, то напечатайте число 2, иначе если одинаковы 3 и 2 числа, то напечатайте число 3, иначе если одинаковы 3 числа, то напечатайте число 4, иначе если

одинаковы 2 и 2 числа, то напечатайте число 5, иначе если одинаковы 2 числа, то напечатайте 6, иначе напечатайте число 7.

12. Напишите программу, удаляющую «крайние» элементы списка.

13. Напишите программу, меняющую местами «крайние» элементы списка.

14. Напишите программу, меняющую местами пару элементов списка по их указанным индексам.

15. Напишите программу, добавляющую в начало списка копию его предпоследнего элемента.

16. Напишите программу, удваивающую первый элемент списка (путём создания копии), если его значение меньше значения второго элемента.

17. Напишите программу, объединяющую первую «половину» элементов первого списка со второй «половиной» элементов второго списка.

18. Напишите программу, объединяющую три заданных списка в один, после предварительного удаления из них последних элементов.

19. Напишите программу, объединяющую два заданных списка, если сумма максимального элемента первого списка и минимального элемента второго списка равна заданному числу; в противном случае возвращается пустой список.

20. Напишите программу, конструирующую список из двух заданных таким образом, что из первого списка берутся только отрицательные числа, а из второго – только положительные.

21. Напишите программу, конструирующую список путём утраивания его каждого элемента.

22. Дан список целых чисел. Упорядочьте по возрастанию только положительные числа.

23. Дан список целых чисел. Упорядочьте по возрастанию только элементы с четными порядковыми номерами в списке.

24. Даны два упорядоченных по не возрастанию списка. Объедините их в новый упорядоченный по не возрастанию список.

25. Если в списке есть отрицательные элементы, найти наибольший из них.

26. Найти все числа, каждое из которых встречается в двух заданных списках.

27. В заданном списке определить максимальное количество подряд идущих положительных элементов, не прерываемых ни нулями, ни отрицательными элементами.

28. Даны два целочисленных списка, содержащих одинаковое количество чисел. Написать программу, которая печатает «Да», если эти списки состоят из одинаковых элементов, и «Нет» – в противном случае.

29. В заданном списке, состоящем из целых чисел, вычислить сумму модулей элементов списка, расположенных после первого отрицательного элемента.

30. В заданном списке, состоящем из целых чисел, найдите сумму и количество элементов списка, попавших в интервал $[a; b]$.

31. Вывести в порядке возрастания цифры, входящие в десятичную запись натурального числа N .

32. Из заданного списка удалить все повторяющиеся элементы (дубликаты) так, чтобы каждое значение встречалось в списке только один раз.

33. В заданном списке целых чисел найти (посчитать) количество положительных и количество отрицательных элементов.

34. Определить количество элементов в заданном списке целых чисел, отличающихся от минимального на 5.

35. Написать программу, которая сжимает серии списка, состоящего из единиц и нулей по следующему принципу:

например, список $[0,0,0,1,1,1,1,1,1,0,0,1,1,1]$ преобразуется в $[4,7,2,4]$ (т. к. начинается с нуля, то сразу записывается количество элементов первой серии);

а список $[1,1,1,0,0,0,0,0,0]$ преобразуется в $[0,3,7]$ (т. к. первая серия – это единицы, то первый элемент преобразованного списка 0).

36. Задан упорядоченный по возрастанию целочисленный список. Сформировать второй список из случайных целых чисел, которые не встречаются в первом списке, но имеют величину больше минимального и меньше максимального из чисел первого списка.

37. Найдите наименьший четный элемент списка. Если такого нет, то выведите первый элемент.

38. Дан список, состоящий только из 0 и 1. Определить самое большое количество подряд идущих единиц и вывести на экран индексы начала и конца этого диапазона.

39. Для списка целых чисел вычислить произведение первого, третьего и шестого положительных элементов и определить их номера в списке.

40. Дан список из 10 элементов. Первые 4 упорядочить по возрастанию, последние 4 – по убыванию.

41. Даны два целочисленных списка с различным количеством элементов.

42. Перераспределить их элементы так, чтобы в первом списке были наименьшие элементы из двух списков, а во втором – наибольшие.

43. Даны два пятизначных числа, необходимо найти количество совпадений по две одинаковые цифры в равносильных разрядах чисел, а также количество совпадений по две одинаковые цифры в различных разрядах этих чисел. Цифра, которая уже участвовала в одной паре совпадения, не учитывается повторно.

44. Например, даны числа 12345 и 27376. Количество совпадений одинаковых цифр в равносильном разряде равно 1 (это цифра 3), количество совпадений одинаковых цифр в различных разрядах равно 1 (это цифра 2).

Вариант	№ задания
1	2, 42, 37
2	4, 44, 39
3	6, 1, 41
4	8, 3, 43
5	10, 5, 17
6	12, 7, 22
7	14, 9, 19
8	16, 11, 27
9	18, 13, 24
10	20, 15, 34
11	22, 17, 25
12	24, 19, 6
13	26, 21, 3
14	28, 23, 12
15	30, 25, 1
16	32, 27, 20
17	34, 29, 8
18	36, 31, 14
19	38, 33, 30
20	40, 35, 32

Лабораторная работа № 9. МНОЖЕСТВА

Цель работы – познакомиться с основными конструкциями, используемыми для работы с множеством; приемами программной реализации на языке программирования Python; произвести отладку и тестирование полученных программ.

Задание:

1. Изучить краткие теоретические сведения.
2. Разработать алгоритмы для решения задач (блок-схемы).
3. Записать алгоритмы на языке программирования Python.
4. Произвести отладку и тестирование программы.
5. Оформить отчет по лабораторной работе.

Отчет о лабораторной работе должен содержать следующие сведения:

- 1) название и цель работы;
- 2) формулировку задачи, схему алгоритма, программный код и результаты решения задачи;
- 3) вывод по работе в целом.

Краткие теоретические сведения

Множеством в языке программирования Python называется неупорядоченная совокупность уникальных значений. В качестве элементов этого набора данных могут выступать любые неизменяемые объекты, такие как числа, символы, строки. В отличие от массивов и списков, порядок следования значений не учитывается при обработке его содержимого. Над одним, а также несколькими множествами можно выполнять ряд операций, благодаря функциям стандартной библиотеки языка программирования Python.

Создание

Перед тем как начать работу с множеством, необходимо для начала его создать. Сделать это можно, просто присвоив переменной последовательность значений, выделив их фигурными скобками. Следующий пример показывает код, в котором создается множество целых чисел под названием `a`, после функция `print` выводит на экран его содержимое

```
a = {1, 2, 0, 1, 3, 2}
print(a)
{0, 1, 2, 3}
```

Как можно заметить, все элементы полученной последовательности являются уникальными, без повторений.

Существует и другой способ создания множеств, который подразумевает использование метода **set**. Аргументом этой функции может быть набор неких данных или даже строка с текстом, как это показано в следующем примере.

```
a = set('data')  
print(a)
```

```
{'d', 'a', 't'}
```

В результате выполнения этого кода, программа заполняет новое множество уникальными символами из входной строки. Содержимое набора также выводится на экран.

Использование

Обычно используется для следующих операций:

- ♦ Проверка, есть ли данное значение в множестве. Для этого используется **in**.

```
a = {0, 1, 2, 3}  
print(2 in a)  
True
```

- ♦ Наоборот, проверка отсутствия. Используется **not in**.

```
a = {0, 1, 2, 3}  
print(2 not in a)
```

```
False
```

- ♦ Перебор всех элементов.

```
for a in {0, 1, 2}:  
    print(a)
```

```
0  
1  
2
```

Генератор

Для создания множества можно в Python воспользоваться генератором, позволяющих заполнять списки, а также другие наборы данных с учетом неких условий.

Следующий код демонстрирует генерацию множества `a` с циклом `for` для нескольких чисел.

```
a = {i for i in [1, 2, 0, 1, 3, 2]}
print(a)

{0, 1, 2, 3}
```

Как и прежде, метод `print` показывает содержимое полученного набора значений, где нет повторений, поскольку одинаковые числа автоматически удаляются.

Изменение множеств

Для управления содержимым множеств в языке Python присутствуют специальные методы, дающие возможность добавлять и удалять отдельные элементы.

Получение размера

Узнать точное количество элементов, входящих в состав множества, поможет метод `len`, принимающий в качестве аргумента набор данных. Функция `print` выводит результат.

```
a = {0, 1, 2, 3}
print(len(a))

4
```

Добавление элемента

Чтобы внести новые значения, потребуется вызывать метод `add`. Аргументом в данном случае будет добавляемый элемент последовательности. В примере кода на Python добавим в множество элемент со значением 4.

```
a = {0, 1, 2, 3}
a.add(4)
print(a)

{0, 1, 2, 3, 4}
```

Удаление элемента

Для удаления элементов из множества используются следующие функции в Python (кроме очистки, которая будет рассмотрена ниже):

- ♦ `remove` – удаление элемента с генерацией исключения в случае, если такого элемента нет;
- ♦ `discard` – удаление элемента без генерации исключения, если элемент отсутствует;
- ♦ `pop` – удаление первого элемента, генерируется исключение при попытке удаления из пустого множества.

Избавиться от лишних значений в наборе данных с помощью `remove`. В качестве входного параметра здесь выступает элемент, который нужно удалить (в примере удалим число со значением 3).

```
a = {0, 1, 2, 3}
a.remove(3)
print(a)
```

```
{0, 1, 2}
```

По поводу функции `pop` хотелось бы отметить, что так как множества не упорядочены, то удалится случайный элемент, который будет находиться в памяти первым. Но если они в памяти хранятся в отсортированном виде, что не факт, то скорее всего будет удален элемент с наименьшим значением. Но на это не стоит рассчитывать.

Полная очистка

Иногда необходимо полностью убрать все элементы. Чтобы не удалять каждый элемент отдельно, используется метод `clear`, не принимающий аргументов. Если вывести содержимое после этой операции, на экране появится только его название.

```
a = {0, 1, 2, 3}
a.clear()
print(a)
```

```
set()
```

В результате получили пустое множество.

Сортировка

Порядок следования элементов не учитывается. Поэтому нет смысла говорить о сортировке множеств в Python 3.

Но с другой стороны все дело обстоит не со всем так. Для быстрого поиска элемента, желательно их хранить в памяти в упорядоченном виде.

В начале рассмотрим, что будет с элементами разных типов данных в одном множестве. Такие элементы не должны сортироваться. Если мы будем выводить элементы с помощью команды `print`, то они выводятся примерно следующим образом:

```
a = {0, 1, 12, 'b', 'ab', 3, 2, 'a'}  
print(a)
```

```
{0, 1, 'b', 3, 2, 12, 'ab', 'a'}
```

Как видим, у нас вывелись не отсортированные значения, если повторить запуск, то порядок будет меняться. Но это только в том случае, если перемешаны элементы разного типа.

Посмотрим, что будет, если попытаемся вывести только числа:

```
a = {0, 1, 12, 3, 2}  
print(a)
```

```
{0, 1, 2, 3, 12}
```

Все элементы выведены упорядоченно. Теперь посмотрим что будет если преобразовать в список:

```
a = {0, 1, 12, 3, 2}  
b = list(a)  
print(b)
```

```
[0, 1, 2, 3, 12]
```

Аналогично, в список значения записались отсортированными по возрастанию.

Операции над множествами

Помимо различных манипуляций с элементами множеств, существуют еще и операции над ними, позволяющие одной строчкой кода выполнять сложные преобразования.

Рассмотрим операции с множествами доступные в Python 3.

Объединение

Чтобы объединить все элементы двух разных множеств, стоит воспользоваться методом `union` на одном из объектов. Следующий пример демонстрирует работу данной функции, где создается последовательность чисел под именем `c`.

```
a = {0, 1, 2, 3}
b = {4, 3, 2, 1}
c = a.union(b)
print(c)
```

```
{0, 1, 2, 3, 4}
```

Добавление

Чтобы добавить все элементы из одного множества к другому, необходимо вызывать метод `update` на первом объекте. Таким образом можно перенести уникальные данные из одного набора чисел в другой, как это показано в следующем примере.

```
a = {0, 1, 2, 3}
b = {4, 3, 2, 1}
a.update(b)
print(a)
```

```
{0, 1, 2, 3, 4}
```

Пересечение

Чтобы найти общие элементы для двух разных множеств, следует применить функцию `intersection`, принимающую в качестве аргумента один из наборов данных. Код, приведенный ниже, создает новую последовательность чисел из пересечения двух множеств в Python 3.

```
a = {0, 1, 2, 3}
b = {4, 3, 2, 1}
c = a.intersection(b)
print(c)

{1, 2, 3}
```

Разность

Чтобы вычислить разность для двух разных множеств, необходимо воспользоваться методом `difference`. Функция позволяет найти элементы, уникальные для второго набора данных, которых в нем нет. Следующий код демонстрирует эту операцию.

```
a = {0, 1, 2, 3}
b = {4, 3, 2, 1}
c = a.difference(b)
print(c)

{0}
```

Отношения между множествами

Для определения подмножеств и надмножеств существуют специальные функции, возвращающие `True` или `False` в зависимости от результата выполнения.

Определение подмножества

Чтобы выяснить, является ли множество `a` подмножеством `b`, стоит попробовать вывести на экран результат выполнения метода **`issubset`**, как в следующем примере. Так как не все элементы набора чисел `a` присутствуют в `b`, функция вернет `False`.

```
a = {0, 1, 2, 3, 4}
b = {3, 2, 1}
print(a.issubset(b))

False
```

Определение надмножества

Чтобы узнать, является ли множество *a* надмножеством *b*, необходимо вызвать метод **issuperset** и вывести результат его работы на экран. Поскольку все элементы набора чисел *b* присутствуют в *a*, функция возвращает **True**.

```
a = {0, 1, 2, 3, 4}
b = {3, 2, 1}
print(a.issuperset(b))

True
```

Тупой замороженный набор

Множество, содержимое которого не поддается изменению имеет тип **frozenset**. Значения из этого набора нельзя удалить, как и добавить новые. В следующем примере демонстрируется создание при помощи стандартной функции.

```
a = frozenset({"осень", "листва"})
print(a)

frozenset({'осень', 'листва'})
```

Поскольку содержимое **frozenset** должно всегда оставаться статичным, перечень функций, с которыми такое множество может взаимодействовать, имеет ограничения.

Преобразование множества

Иногда возникает необходимость представления уже готовой последовательности значений в качестве совсем другого типа данных. Возможности языка позволяют конвертировать любое множество в строку, словарь или список при помощи стандартных функций.

Строка

Для преобразования множества в строку используется конкатенация текстовых значений, которую обеспечивает функция **join**. В этом случае ее аргументом является набор данных в виде нескольких строк. Запятая в кавычках выступает в качестве символа, разделяющего значения. Метод **type** возвращает тип данных объекта в конце приведенного кода.

```

a = {'set', 'str', 'dict', 'list'}
b = ','.join(a)
print(b)
print(type(b))

set,dict,list,str
<class 'str'>

```

Словарь

Чтобы получить из множества словарь, следует передать функции **dict** набор из нескольких пар значений, в каждом из которых будет находиться ключ. Метод **print** демонстрирует на экране содержимое полученного объекта, а **type** отображает его тип.

```

a = {'a', 2), ('b', 4)}
b = dict(a)
print(b)
print(type(b))

{'b': 4, 'a': 2}
<class 'dict'>

```

Следует отметить, что каждый элемент для такого преобразования – кортеж состоящий из двух значений: ключ будущего словаря; значение, соответствующее ключу.

Список

По аналогии с предыдущими преобразованиями можно получить **список** неких объектов. На этот раз используется метод **list**, получающий в качестве аргумента множество **a**. На выходе функции **print** отображаются уникальные значения для изначального набора чисел.

```

a = {1, 2, 0, 1, 3, 2}
b = list(a)
print(b)
print(type(b))

[0, 1, 2, 3]
<class 'list'>

```

Следующая таблица отображает краткую сводку по всем пройденным функциям, включая их названия и назначение.

Название	Назначение
len	Получение размера
add	Добавление элемента
remove	Удаление элемента
clear	Очистка
union	Объединение
update	Добавление всех элементов одного множества в другое
intersection	Нахождение множества, элементы которого находятся на пересечении двух множеств
difference	Нахождение множества, элементы которого входят в первое, но не входят во второе множество
issubset	Проверка, является ли множество подмножеством
issuperset	Проверка, является ли множество надмножеством

Пример 1

Постановка задачи. Написать программу, которая выводит все цифры, встречающиеся в символьной строке больше одного раза.

Метод решения. Входная строка может содержать цифры, пробелы и латинские буквы. Программа должна вывести в одну строчку в порядке возрастания все цифры, встречающиеся во входной строке больше одного раза. Если таких цифр нет, нужно вывести слово 'NO'.

Текст программы

```

1 a = set()
2 st=input()
3 b=set()
4 for x in st:
5     if '0'<=x<='9' and x not in a:
6         a.add(x)
7     elif x in a:
8         b.add(x)
9 if len(b)>0:
10     b=sorted(list(b))
11     print(*b,sep='')
12 else: print('NO')
```

Результат

```

t1y2u3i4o5
NO

In [17]: run

ab1n3zk2
2

In [18]:
```

Пример 2

Постановка задачи. Написать программу, которая удаляет из строки все повторяющиеся символы.

Метод решения. На вход программы подаётся строка, содержащая символы таблицы ASCII. Программа должна вывести исходную строку, из которой удалены все повторяющиеся символы.

Текст программы

```
1 a = input()
2 b=set()
3 s=''
4 for x in a:
5     if x not in b:
6         b.add(x)
7         s+=x
8 print(s)
9
```

Результат

```
In [18]: runf:
abc13a1b2z3c
abc132z

In [19]: runf:
Смоляк
Смоляк
```

Задачи для самостоятельной работы

1. Из множества целых чисел 1...20 выделить:
 - а) множество чисел, делящихся на 6 без остатка;
 - б) множество чисел, делящихся без остатка или на 2, или на 3.
2. Определить количество различных гласных букв латинского алфавита в некотором тексте.

3. В заданной последовательности литер, состоящей из букв латинского алфавита, определить общее число вхождений в нее букв «а», «е», «с», «h».

4. Переменные x , y , z – множества. Переменной x присвоить множество всех целых чисел от 8 до 22, переменной y – множество всех простых чисел от 8 до 22, а переменной z – множество всех составных чисел из этого же диапазона.

Примечание: где возможно, использовать операторы, предназначенные для работы со множествами.

5. Дан текст. В алфавитном порядке напечатать (по разу) все строчные русские гласные буквы (а, е, о, у, ы, э, ю, я, и, ё), входящие в этот текст.

6. Дан текст из строчных латинских букв. Напечатать:
 - а) первые вхождения букв в текст, по возможности, сохраняя их исходный взаимный порядок;
 - б) все буквы, входящие в текст не менее двух раз;
 - в) все буквы, входящие в текст по одному разу.

7. В возрастающем порядке напечатать все целые числа из диапазона 1..10000, представимые в виде $n*m+m*m$, где $n, m \geq 0$. Для представления диапазона использовать множество.

8. Подсчитать общее количество цифр и знаков «+», «-», и «*», входящих в строку s.

9. Даны целые числа от 1 до 50. Определить, сколько среди них чисел Фибоначчи и сколько чисел, первая значащая цифра в десятичной записи которых 2 или 3.

10. Дано множество M, содержащее числа от 0 до 99. Подсчитать количество элементов во множестве A из M.

11. Подсчитать количество различных (значащих) цифр в десятичной записи натурального числа n.

12. Напечатать, желательно, в возрастающем порядке все цифры, не входящие в десятичную запись натурального числа n.

13. В порядке убывания напечатать все целые числа из диапазона 1..10000, которые представимы в виде $n^2 + 2k^2$, но не представимы в виде $7ij + j + 3$ ($n, k, i, j \geq 0$).

14. Дано множество продуктов $product = \{\text{«хлеб»}, \text{«масло»}, \text{«молоко»}, \text{«мясо»}, \text{«рыба»}, \text{«соль»}, \text{«сыр»}, \text{«колбаса»}, \text{«сахар»}, \text{«чай»}, \text{«кофе»}\}$. Есть список магазинов, каждый элемент которого – множество продуктов в этом магазине.

Определить значения множеств A, B, C: A – множество продуктов, которые есть во всех магазинах; B – множество продуктов, каждый из которых есть хотя бы в одном магазине; C – множество продуктов, которых нет ни в одном магазине.

15. Дано множество имен = $\{\text{«Вася»}, \text{«Володя»}, \text{«Ира»}, \text{«Лидида»}, \text{«Марина»}, \text{«Миша»}, \text{«Наташа»}, \text{«Олег»}, \text{«Оля»}, \text{«Света»}, \text{«Юля»}\}$. Есть список, каждый элемент которого – множество людей, побывавших у какого-либо члена группы. Определить, есть ли хотя бы один человек, побывавший в гостях у всех остальных из группы.

16. Дано множество городов $Town = \{\text{«a»}, \text{«b»}, \text{«c»}, \text{«d»}, \text{«e»}, \text{«f»}, \text{«g»}, \text{«h»}\}$. Есть список, каждый элемент которого – множество городов, в которые можно попасть из города с заданным номером из множества Town. Определить K – множество городов, в которые можно попасть автобусом (за один рейс или через другие города) из города H.

17. Дан текст из цифр и строчных латинских букв. Определить, каких букв – гласных или согласных – больше в этом тексте.

18. Проверить введенную пользователем строку на наличие недопустимых символов. В качестве первого символа допустимы

только буквы и знак подчеркивания. Остальные символы могут быть буквами, цифрами и знаком подчеркивания.

19. Известны марки машин, изготавливаемые в данной стране и импортируемые за рубеж. Даны некоторые N стран. Определить для каждой из марок, какие из них были:

- а) доставлены во все страны;
- б) доставлены в некоторые из стран;
- в) не доставлены ни в одну страну.

20. В классе учатся n учеников. Известны их имена. Известно так же, кто был в гостях у Кати, кто был у Васи и т. д. Определить, есть ли в классе хотя бы один человек, который не был в гостях ни у кого из своих одноклассников. Если есть такие ученики, то вывести их имена, если нет, то сообщить об этом.

21. Дана строка символов. Определить количество различных символов, которые являются буквами или цифрами, вывести их на печать, используя множества.

22. Дана последовательность символов. Требуется построить и напечатать множество, элементами которого являются встречающиеся в последовательности знаки препинания.

23. Дана последовательность символов. Требуется построить и напечатать множество, элементами которого являются встречающиеся в последовательности цифры от 1 до 3 и числа от 17 до 99 включительно.

Лабораторная работа № 10. ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ

Цель работы – получить первоначальные навыки создания и использования функций; приемы программной реализации на языке программирования Python. Произвести отладку и тестирование полученных программ.

Задание:

1. Изучить краткие теоретические сведения.
2. Разработать алгоритмы для решения задач (блок-схемы).
3. Записать алгоритмы на языке программирования Python.
4. Произвести отладку и тестирование программы.
5. Оформить отчет по лабораторной работе.

Отчет о лабораторной работе должен содержать следующие сведения:

- 1) название и цель работы;
- 2) формулировку задачи, схему алгоритма, программный код и результаты решения задачи;
- 3) вывод по работе в целом.

Краткие теоретические сведения

Функция – это средство (способ) группирования фрагментов программного кода таким образом, что этот программный код может вызываться многократно с помощью использования имени функции.

Использование функций в программах на **Python** дает следующие взаимосвязанные преимущества:

- ✓ избежание повторения одинаковых фрагментов кода в разных частях программы;
- ✓ уменьшение избыточности исходного кода программы. Как следствие, уменьшение логических ошибок программирования;
- ✓ улучшенное восприятие исходного кода программы в случаях, где вместо блоков многочисленных инструкций (операторов) вызываются имена готовых протестированных функций. Это, в свою очередь, также уменьшает количество ошибок;
- ✓ упрощение внесения изменений в повторяемых блоках кода, организованных в виде функций. Достаточно внести изменения

только в тело функции, тогда во всех вызовах данной функции эти изменения будут учтены;

- ✓ с помощью функций удобно разбивать сложную систему на более простые части. Значит, функции – удобный способ структурирования программы;

- ✓ уменьшение трудозатрат на программирование, а, значит, повышение производительности работы программиста.

В языке программирования **Python** для функций характерны следующие особенности:

- функции могут возвращать результат;
- функции могут получать входные параметры, влияющие на полученный результат.

Инструкция def. Общая форма объявления функции в Python. Создание функции

Чтобы функцию можно было использовать в программе, ее нужно создать (объявить).

В языке **Python** создание (или объявление) функции выполняется с помощью инструкции **def**. Если интерпретатор достигает инструкции **def**, он создает новый объект функции и связывает упакованный код функции с ее именем.

Функция может возвращать или не возвращать значение. Если функция не возвращает значения, то считается, что она автоматически возвращает объект **None**, который игнорируется. В этом случае инструкция **return** необязательна. Общая форма такой функции следующая:

```
def <name>(arg1, arg2, ..., argN):  
    <statement1>  
    <statement2>  
    ...  
    <statementN>
```

где **name** – имя функции, с которым будет связан объект функции и список параметров (**arg1, arg2, ..., argN**). Объект функции получает имя **name** и создается во время выполнения (в Python нет времени компиляции);

arg1, arg2, argN – список параметров, которые может получать функция. Функция также может быть без параметров;

statement1, statement2, statementN – одна или несколько инструкций, которые следуют с отступлениями относительно **def**.

Если функция возвращает значение, то ее общая форма следующая

```
def <name>(arg1, arg2, ..., argN):  
    <statement1>  
    <statement2>  
    ...  
    <statementN>  
    return <value>
```

здесь **name** – имя функции;

arg1, arg2, ..., argN – параметры функции (если такие есть);

statement1, statement2, statementN – инструкции, которые реализованы в теле функции;

value – значение, возвращаемое функцией.

Определение функции с помощью инструкции **def** может быть реализовано внутри других инструкций, например, внутри инструкции **if**.

Вызов функции

После того, как функция создана (объявлена), можно осуществлять ее вызов. Вызов функции осуществляется с помощью ее имени и перечня параметров.

Например. Пусть объявлена следующая функция

```
def Product(a, b):  
    return a*b
```

тогда ее вызов может быть, например, таким

```
res = Product(7,9)
```

```
def Product(a, b):  
    return a*b  
res = Product(7,9)  
print(res)
```

In [10]: run
63
In [11]: |

Интерпретатор, встретив инструкцию **def** выполняет следующие действия:

- создает новый объект функции с именем **Product**;
- упаковывает программный код функции и связывает объект с именем **Product**;
- вызывает функцию в программе.

Если указать **res = Product('abc',3)**, получим следующий результат:

```
def Product(a, b):  
    return a*b  
res = res = Product('abc',3)  
print(res)
```

```
In [11]: run  
abcabcabc  
  
In [12]:
```

Пример объявления и использования функции без параметров

Ниже приведен пример объявления и использования функции, которая не содержит параметров. Функция носит имя **Privet** и выводит на экран текст «Привет! Как дела?» 3 раза.

```
6 def Privet():  
7     print("Привет! Как дела?")  
8 i=0  
9 while i<3:  
10     Privet()  
11     i=i+1  
12
```

```
Привет! Как дела?  
Привет! Как дела?  
Привет! Как дела?  
  
In [13]: |
```

Пример объявления и использования функции, которая получает 1 параметр

Ниже приведен пример создания и использования функции **Print1()**, которая получает один параметр и выводит его на экран 3 раза.

```
def Print1(t):  
    i=0  
    while i<3:  
        print(t)  
        i=i+1  
  
Print1('Программирование Python')  
Print1(2020)  
Print1(True)
```

```
Программирование Python  
Программирование Python  
Программирование Python  
2020  
2020  
2020  
True  
True  
True
```

Как видно из результата, в языке **Python** тип параметра, который передается в функцию, может быть **любым**. Интерпретатор распознает его по объекту, который передается в качестве параметра.

*Пример объявления и использования функции,
которая получает параметр и возвращает результат*

Ниже приводится пример функции, которая получает параметр и возвращает результат. Функция осуществляет реверсирование строки.

Функция, реверсирующая строку

```
def Reverse(s):  
    s2 = ''  
    i = len(s)-1  
    while i>=0:  
        s2 = s2 + s[i]  
        i = i-1  
    return s2  
  
# Вызов функции Reverse()  
s1 = "ПРОГРАММИРОВАНИЕ"  
s2 = Reverse(s1)  
print("s2 = ", s2)
```

```
In [15]: runfile('C:/Use  
s2 = ЕИНАВОРИММАРГОРП  
  
In [16]:
```

*Пример объявления функции в инструкции if.
Реализация альтернативы*

Определение функции с помощью средства **def** может быть вложено в другую инструкцию, например, инструкцию **if**. В этом случае возникает альтернатива при выполнении программы.

Пример. В примере на основании введенного значения **n** создается функция **Fn()**, которая вычисляет тригонометрическое выражение по следующему правилу:

- $\sin(x) \cdot \cos(x)$, если $n==1$;
- $\sin(x^2) + \cos(x)$, если $n==2$;
- $1 - \sin(x)$, в противном случае.

Затем эта функция вызывается для заданного значения x .

```
# По заданному значению n определить функцию Fn(),  
# которая вычисляет:  
# sin(x)*cos(x), если n==1  
# sin(x*x)+cos(x), если n==2  
# 1-sin(x), в противном случае
```

```

import math
n = int(input('n = '))
x = float(input('x = '))

# Определяем код функции Fn()
if (n==1):
    def Fn(x):
        print("n==1. res = sin(x)*cos(x).")
        return math.sin(x)*math.cos(x)
else:
    if (n==2):
        def Fn(x):
            print("n==2. res = sin(x*x)+cos(x)")
            return math.sin(x*x)+math.cos(x)
    else:
        def Fn(x):
            print("n==3. res = 1-sin(x)")
            return 1-math.sin(x)

# Вызов функции Fn()
res = Fn(x)
print("res = ", res)

```

```

n = 3
x = 1.2
n==3. res = 1-sin(x)
res = 0.06796091403277371

In [17]: |

```

Динамическая типизация. В языке **Python** все данные представлены объектами. Объекты определяют синтаксическое содержание операции. Параметры функции есть объекты, которые могут быть разных типов. Параметры функции используются в операциях, которые могут оперировать разными типами. В языке Python этот процесс называется *динамической типизацией*.

Например, операция ***** может применяться как к числам, так и к строкам. В случае с числами эта операция означает умножение. В случае со строками эта операция позволяет повторить строку заданное количество раз.

Пример 1. Демонстрируется функция **Mult()**, которая получает два параметра **a, b**. Для полученных параметров выполняется операция *****.

Вызов функции **Mult()** для разных типов объектов:

```

def Mult(a,b):
    return a*b

x = 8
y = 5
z = Mult(x,y)
print(z)
s1 = "Привет! "
n = 3
s2 = Mult(s1,n)
print(s2)

```

```

40
Привет! Привет! Привет!

In [19]: |

```

Пример 2. Демонстрируется функция **Sum ()**, которая получает два параметра **a, b**. Функция возвращает результат операции + над параметрами.

Вызов функции **Sum()** для разных типов объектов:

```
def Sum(a,b):  
    return a+b  
x = 255  
y = 552  
z = Sum(x,y)  
print(z)  
print()  
s1 = "Информационные системы "  
s2 = "и технологии!"  
s3 = Sum(s1,s2)  
print(s3)
```

```
In [22]: runfile('C:/Users/AMakhnenko/ur  
807
```

Информационные системы и технологии!

```
In [23]: |
```

Использование локальных переменных внутри функций

Глобальные переменные — это переменные, объявленные в программе вне функции. В Python глобальные переменные *видны в любой части модуля*, включая функции.

Внутри функции могут быть объявлены локальные переменные.

Локальная переменная — это имя (объект), которое доступно только внутри определения функции (инструкции def). Локальная переменная существует только во время выполнения функции.

В теле любой функции локальные переменные определяются на основе следующих признаков:

- ♦ если переменная размещается в левой части операции присваивания;

- ♦ если переменная (имя) есть параметр (аргумент) функции;

- ♦ если переменная есть итератором в цикле for.

Пример. В примере создается функция **SquareTriangle()**, которая вычисляет площадь треугольника по его сторонам. Длины сторон треугольника с именами **a, b, c** есть входными параметрами функции. Если с **a, b, c** невозможно образовать треугольник, то функция возвращает значение **None**.

Затем происходит вызов функции **SquareTriangle()** и обработка результата возврата из функции.

```

import math
def SquareTriangle(a,b,c):
    if (((a+b)<c) or ((b+c)<a) or ((a+c)<b)):
        return None
    else:
        p = (a+b+c)/2 # это локальная переменная p
        s = math.sqrt(p*(p-a)*(p-b)*(p-c)) # локальная переменная s
        return s

# Использование функции SquareTriangle()
s = SquareTriangle(5,4,5)

if (s==None):
    print("The values of a, b, c is incorrect.")
else:
    print("s = ", s)

import math
def SquareTriangle(a,b,c):
    if (((a+b)<c) or ((b+c)<a) or ((a+c)<b)):
        return None
    else:
        p = (a+b+c)/2 # это локальная переменная p
        s = math.sqrt(p*(p-a)*(p-b)*(p-c)) # локальная переменная s
        return s

# Использование функции SquareTriangle()
s = SquareTriangle(7,3,2)

if (s==None):
    print("The values of a, b, c is incorrect.")
else:
    print("s = ", s)

```

```

In [24]: runfile('C:/U:
s = 9.16515138991168

```

```

In [25]:

```

```

The values of a, b, c is incorrect.

```

```

In [26]:

```

Задачи для самостоятельной работы

1. Даны действительные числа a_0, \dots, a_6 . Получить для $x = 1, 2, 3, 4$ значения $p(x+1)-p(x)$, где: $p(y) = a[1]*y^6 + a[2]*y^5 + a[3]*y^4 + a[4]*y^3 + \dots + a[0]$.

2. Даны действительные числа s, t, a_0, \dots, a_{12} . Получить $p(1)-p(t)+p(st)*p(s-t)-p(1)*p(1)$, где $p(x) = a_{12}x^{12} + a_{11}x^{11} + \dots + a_0$.

3. Дано натуральное число n . Среди чисел $1, 2, \dots, n$ найти все те, которые можно представить в виде суммы квадратов двух натуральных чисел, определив функцию, позволяющую распознавать полные квадраты.

4. Найти значение переменной z , заданной суммой функций:

$$z = f(a,b) + f(a^2,b^2) + f(a-1,b) + f(a-b,b) + f(a^2+b^2,b^2-1),$$

$$\text{где: } f(u, t) = \begin{cases} u^2 + t^2, & \text{если } u > 0, t > 0; \\ u + t^2, & \text{если } u \leq 0, t \leq 0; \\ u - t, & \text{если } u > 0, t \leq 0; \\ u + t, & \text{если } u \leq 0, t > 0. \end{cases}$$

5. Найти значение переменной z , заданной суммой функций:
 $z = f(\sin b, a) + f(\cos b, a) + f(\sin b, a-1) + f(\sin b - \cos b, a \cdot a - 1) + f(\sin b \cdot \sin b - 1, \cos a + a)$,

$$\text{где: } f(u, t) = \begin{cases} u + \sin t, & \text{если } u > 0; \\ u + t, & \text{если } u \leq 0. \end{cases}$$

6. Найти значение переменной z , заданной суммой функций:
 $z = f(|x|, y) + f(a, b) + f(|x|+1, -y) + f(|x|-|y|, x) + f(x+y, a+b)$,

$$\text{где: } f(u, t) = \begin{cases} u + 2t, & \text{если } u \geq 0; \\ u + t, & \text{если } u \leq -1; \\ u^2 - 2t + 1, & \text{если } -1 < u < 0. \end{cases}$$

7. Найти значение переменной z , заданной суммой функций:
 $z = f(\sin x + \cos y, x+y) + f(\sin x, \cos y) + f(x-y, x) + f(\sin x - 2, a) + f(a+3, b+1)$,

$$\text{где: } f(u, t) = \begin{cases} u + t, & \text{если } u > 1; \\ u - t, & \text{если } 0 \leq u \leq 1; \\ t - u, & \text{если } u < 0. \end{cases}$$

8. Натуральное число называется палиндромом, если оно читается одинаково с обеих сторон (например, 171). Возьмем произвольное натуральное число X . Если оно не палиндром, то перевернем его и сложим с исходным числом. Если сумма не является палиндромом, то сделаем с ней указанные операции. Работу продолжать до тех пор, пока не получится палиндром. На экран вывести полученное число и количество шагов. Для получения перевернутого числа составить функцию.

9. Составить программу вывода разложения бинома Ньютона:
 $(a+b)^n = C(0, n)a^n b^0 + C(1, n)a^{n-1}b^1 + \dots + C(n, n)a^0b^n$,

$$\text{где } C(0, n) = C(n, n) = 1, \quad C(m, n) = \frac{n!}{m!(n-m)!}.$$

10. Переменной t присвоить значение true, если уравнения $x^2 + 6.2x + a^2 = 0$ и $x^2 + ax + b - 1 = 0$ имеют вещественные корни и при этом оба корня первого уравнения лежат между корнями второго, и присвоить значение false во всех остальных случаях.

11. По заданным вещественным числам a_0, \dots, a_{30} , b_0, \dots, b_{30} , c_0, \dots, c_{30} , x , y , z вычислить величину:

$$\frac{(a_0 x^{30} + a_1 x^{29} + \dots + a_{30})^2 - (b_0 y^{30} + b_1 y^{29} + \dots + b_{30})^2}{c_0 (x+z)^{30} + c_1 (x+z)^{29} + \dots + c_{30}}.$$

12. Даны два списка списков по 10 элементов. Каждый элемент списка – это список, состоящий из целых чисел. Вывести тот список, в котором находится список (элемент), содержащий минимальную сумму чисел.

13. Дан список списков A , содержащий M элементов, в каждом из которых N вещественных чисел. Найти величину $x_1x_n + x_2x_{n-1} + \dots + x_nx_1$, где x_i – максимальный элемент i -го элемента списка A .

14. Даны три списка списков, состоящих из 5 элементов, в каждом из которых 4 целых числа. Вывести тот из них, где больше элементов вида $[0, 0, 0, 0]$ (если таких списков несколько, то вывести их все).

15. Даны три списка списков A, B, C , состоящих из 4 элементов, каждый из которых состоит из 5 вещественных чисел. Вычислить величину:

$$\frac{\langle A \rangle + \langle B \rangle + \langle C \rangle}{\langle A + B + C \rangle}, \text{ где:}$$

$$\langle D \rangle = \max_{0 \leq j \leq 4} |D_{0,j}| + \max_{0 \leq j \leq 4} |D_{1,j}| + \max_{0 \leq j \leq 4} |D_{2,j}| + \max_{0 \leq j \leq 4} |D_{3,j}|.$$

16. Сгенерировать десять списков из случайных чисел. Вывести их и сумму их элементов на экран. Найти среди них один с максимальной суммой элементов. Указать какой он по счету, повторно вывести этот список и сумму его элементов. Заполнение списка и подсчет суммы его элементов оформить в виде отдельных функций (стандартную функцию подсчета суммы элементов списка использовать нельзя).

17. Найти наибольшие общие делители (НОД) для списка кортежей, составленных из пар чисел. Для нахождения НОД использовать функцию.

18. Дан список из N элементов, состоящих из N целых чисел. Вычислить сумму элементов главной или побочной диагонали, полученной из этого списка матрицы, в зависимости от выбора пользователя. Сумма элементов любой диагонали должна вычисляться в одной и той же функции.

19. Дан список, состоящий из натуральных чисел. Выполнить сортировку данного списка по возрастанию суммы цифр чисел. Например, дан список чисел $[14, 30, 103]$. После сортировки он будет таким: $[30, 103, 14]$, так как сумма цифр числа 30 составляет 3, числа 103 равна 4, числа 14 равна 5. Вывести на экран исходный

список, отсортированный список, а также для контроля суммы цифр каждого числа отсортированного списка. Для нахождения суммы цифр использовать функцию.

20. Найти средние арифметические пяти списков, состоящих их десяти целых чисел. Для нахождения среднего арифметического использовать функцию.

21. Дан список, состоящий из целых чисел. Написать и протестировать функцию, которая из заданного списка формирует новый список, состоящий только из элементов, дважды входящих в первый список.

22. Дан список, состоящий из целых чисел. С клавиатуры вводится число N. Написать и протестировать функцию, которая сжимает список, удаляя из него элементы, равные числу N.

23. Написать функцию, которая циклически сдвигает список вправо или влево на указанное число позиций. Сдвиг также должен быть кольцевым, то есть те элементы, которые уходят вправо или влево за пределы списка, должны помещаться с другого его конца.

Например, дан список: [1,2,3,4,5,6]. Кольцевой сдвиг вправо на 2 единицы: [5,6,1,2,3,4].

24. Написать функцию, заменяющую подстроку, которая начинается с первого вхождения в строку s открывающей квадратной скобки и заканчивается соответствующей ей закрывающей квадратной скобкой, на строку s1 и возвращающую подстроку, заключенную между скобками в качестве своего значения.

Лабораторная работа № 11. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Цель работы – познакомиться с правилами создания регулярных выражений и получить первоначальные навыки их использования; приемы программной реализации на языке программирования Python. Произвести отладку и тестирование полученных программ.

Задание:

1. Изучить краткие теоретические сведения.
2. Разработать алгоритмы для решения задач (блок-схемы).
3. Записать алгоритмы на языке программирования Python.
4. Произвести отладку и тестирование программы.
5. Оформить отчет по лабораторной работе.

Отчет о лабораторной работе должен содержать следующие сведения:

- 1) название и цель работы;
- 2) формулировку задачи, схему алгоритма, программный код и результаты решения задачи;
- 3) вывод по работе в целом.

Краткие теоретические сведения

Регулярные выражения предназначены для выполнения в строке сложного поиска или замены. В языке Python использовать регулярные выражения позволяет модуль **re**. Прежде чем задействовать функции из этого модуля, необходимо подключить модуль с помощью инструкции:

import re

Найдем строку, чтобы увидеть, начинается ли она с «Доброе» и заканчивается «утро»:

```
import re
txt = "Доброе осеннее утро"
x = re.search("^Доброе.*утро$", txt)
print(x)
```

<re.Match object; span=(0, 19), match='Доброе осеннее утро'>
In [39]:

Если строка не найдена:

```
import re
txt = "Доброе осеннее утро"
x = re.search("^Доброе.*утречко$", txt)
print(x)
```

In [39]: runfile
None
In [40]:

Модуль **re** предлагает набор функций, которые позволяют нам искать строку на предмет соответствия:

Функции	Значение
findall	Возвращает список со всеми совпадениями
search	Возвращает объект Match, если в строке есть совпадение
split	Возвращает список, из строки, которую разделили по шаблону
sub	Заменяет совпадение по шаблону, на заданную строку

Метасимволы – это символы с особым значением:

Символ	Значение	Пример
[]	Содержит символы для поиска вхождений	[a-m]
\	Сигнализирует о специальном символе (также может использоваться для экранирования специальных символов)	\d
.	Любой символ, кроме новой строки (\n)	"he...o "
^	Строка начинается с	"^hello "
\$	Строка заканчивается	"world\$"
*	0 и более вхождений	"aix*"
+	1 и более вхождений	"aix+"
{ }	Указанное количество вхождений	"al{2} "
	Или	"falls stays"
()	Группирует шаблон	

Специальные пары символов представляют собой \, за которым следует один из символов в списке ниже, и имеет специальное значение:

Символ	Значение	Пример
\A	Ищет символы в начале строки	"\AThe"
\b	Ищет символы в начале или конец слова, в зависимости от расположения	r"\bain" r"ain\b"
\B	Ищет символы которые находятся НЕ в начале или конце строки	r"\Bain" r"ain\B"
\d	Ищет совпадения с числами 0-9	"\d"
\D	Ищет совпадение, где строка не содержит числа	"\D"
\s	Ищет совпадение с символом пробела	"\s"
\S	Ищет совпадение, где строка НЕ содержит пробел	"\S"
\w	Ищет совпадение, где строка содержит буквы, цифры или символ подчеркивания (_)	"\w"
\W	Ищет совпадение, где строка НЕ содержит буквы, цифры или символ подчеркивания (_)	"\W"
\Z	Ищет символы в конце строки	"Spain\Z"

```

import re
txt = "The rain in Spain"
x = re.search("\AThe", txt)
y=re.search(r"\bain",txt)
z=re.search( r"ain\b",txt)
t=re.search( "\d",txt)
a=re.search( "\d",txt)
b=re.search( "\s",txt)
bl=re.search( "\S",txt)
f=re.search( "\w",txt)
fl=re.search( "\W",txt)
h=re.search( "Spain\Z",txt)
print(x)
print(y)
print(z)
print(t)
print(a)
print(b)
print(bl)
print(f)
print(fl)
print(h)

```

```

In [15]: runfile('C:/Users/AMakhnenko/Desktop/unt:
<re.Match object; span=(0, 3), match='The'>
None
<re.Match object; span=(5, 8), match='ain'>
<re.Match object; span=(0, 1), match='T'>
None
<re.Match object; span=(3, 4), match=' '>
<re.Match object; span=(0, 1), match='T'>
<re.Match object; span=(0, 1), match='T'>
<re.Match object; span=(3, 4), match=' '>
<re.Match object; span=(12, 17), match='Spain'>

```

In [16]:

Комбинации – это набор символов внутри пары квадратных скобок [] со специальным значением:

Комбинации	Значение
[arn]	Возвращает совпадение, в котором присутствует один из указанных символов (a, r или n)
[a-n]	Возвращает совпадение для с символом нижнего регистра в алфавитном порядке между a и n, включая их
[^arn]	Возвращает совпадение для любого символа, КРОМЕ a, r и n
[0123]	Возвращает совпадение, в котором присутствует любая из указанных цифр (0, 1, 2 или 3)
[0-9]	Возвращает совпадение с любой цифрой от 0 до 9
[0-5][0-9]	Возвращает совпадение с любыми двузначными числами от 0 до 59
[a-zA-Z]	Возвращает совпадение с любым символом английского алфавита между a и z, включая строчные буквы и прописные
[а-яА-ЯёЁ]	Возвращает совпадение с любым символом русского алфавита между а и я, включая строчные буквы и прописные
[+]	В комбинациях символы +, *, ., , (), \$, {} не имеют особого значения, поэтому [+]: будет искать любой + в строке

Метод re.search() принимает на вход шаблон и строку и возвращает объект match в случае успеха, а в противном случае, если

шаблон не был найден, возвращает None. Объект match имеет метод group(), который возвращает сам шаблон.

Пример:

```
import re
s = "Мой номер 123"
a = re.search(r'\d\d\d', s)
print(a)
```

<re.Match object; span=(10, 13), match='123'>

```
import re
s = "Мой номер 123"
a = re.search(r'\d\d\d', s)
b=a.group()
print(b)
```

123

В примере выше использован шаблон \d\d\d. Шаблон \d в регулярных выражениях сравнивает единичные цифры, следовательно \d\d\d будет сравнивать цифры вида 111, 222, 786. Соответственно, цифры вида 12, 1444 будут пропущены.

Пример:

```
import re
s = "e-mail Артема Иванова Ivanart@mail.ru"
a = re.search(r'[\w.-]+@[ \w.-]+' , s)
```

Приведенное выше регулярное выражение будет искать e-mail адрес

```
if a:
    print(a.group())
else:
    print("сравнений не обнаружено")
```

Ivanart@mail.ru

Здесь мы использовали шаблон `[\\w.-]+@[\\w.-]+` чтобы найти в строке соответствие с e-mail адресом. В случае успеха метод `re.search()` возвращает объект **a**, а уже при помощи метода этого объекта `group()` можно извлечь хранящийся там текст. В данном примере это `Ivanart@mail.ru`.

Группировка выражений позволяет нам извлекать части из нужной строки. Создать группу можно при помощи круглых скобок `()`. Допустим, мы хотим извлечь отдельно имя пользователя и имя хоста из e-mail адреса, такого как в предыдущем примере. Для этого нам нужно заключить в разные скобки шаблон для имени и шаблон для хоста следующим образом:

```
a = re.search(r'([\\w.-]+)@([\\w.-]+)', s)
```

Обратите внимание: добавление скобок не означает изменение шаблона для поиска соответствия. Наш код по-прежнему ищет полное совпадение с заданным шаблоном. Если поиск увенчается успехом, **a.group(1)** будет соответствовать содержимому первых круглых скобок, а **a.group(2)** – содержимому вторых круглых скобок.

```
import re
s = "e-mail Артема Иванова Ivanart@mail.ru"
a = re.search('([\\w.-]+)@([\\w.-]+)', s)
if a:
    print(a.group()) ## Ivanart@mail.ru (полное сравнение)
    print(a.group(1)) ## Ivanart (the username, первая группа)
    print(a.group(2)) ## mail (the host, вторая группа)
```

```
Ivanart@mail.ru
Ivanart
mail.ru
```

Функция `findall()` возвращает список, содержащий все совпадения.

В случае успеха данная функция возвращает список, в котором в виде строк содержатся все искомые вхождения по порядку. Если вхождений нет, то эта функция возвратит пустой список.

Пример:

```
import re
string = "The rain in Spain"
x = re.findall("ai", string)
print(x)
```

```
['ai', 'ai']
```

Пример:

```
import re
string = "The rain in Spain"
x = re.findall("Portugal", string)
print(x)
```

```
[]
```

Пример:

```
import re
s = "Tim's phone numbers are 12345-41521 and 78963-85214"
tel = re.findall(r'\d{5}', s)
```

```
if tel:
    print(tel)
```

```
['12345', '41521', '78963', '85214']
```

Функция **findall()** также позволяет использовать группировку выражений. Когда заданы скобочные группы, функция **findall()** возвращает список кортежей, в которых содержатся искомые соответствия.

Пример:

```
import re
s = "Tim's phone numbers are 12345-41521 and 78963-85214"
tel = re.findall(r'(\d{5})-(\d{5})', s)
print(tel)
```

```
for i in tel:
```

```
print()
print(i)
print("Первая группа", i[0])
print("Вторая группа", i[1])
```

```
('12345', '41521')
Первая группа 12345
Вторая группа 41521
('78963', '85214')
Первая группа 78963
Вторая группа 85214
```

Опциональные флаги. Обе функции, `re.search()` и `re.findall()`, могут принимать необязательные параметры – флаги. С их помощью можно слегка изменять поведение функции.

Флаги	Описание
<code>re.IGNORECASE</code>	При поиске соответствий будет игнорироваться регистр символов
<code>re.DOTALL</code>	Позволяет (.) искать символ новой строки. По умолчанию (.) ищет любой символ кроме символа новой строки
<code>re.MULTILINE</code>	Благодаря этому флагу ^ и \$ будут означать начало и конец каждой строчки (line) в многострочной строке (string). По умолчанию эти символы означают начало и конец строки (string)

Пример. Предположим, у нас имеется вот такой многострочный текст:

```
text = """<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Уроки по Python</title>
</head>
<body>
<script type="text/javascript">
let o = document.getElementById('id_div');
```

```
console.log(obj);
</script>
</body>
</html>"""
```

И мы хотим выделить содержимое тега script.

Запишем регулярное выражение:

```
match = re.findall(r"^<script.*?>([w\W]+)(?=</script>)", text,
re.MULTILINE)
```

Получим:

```
["\nlet o = document.getElementById('id_div');\nconsole.log(obj);\n"]
```

Пример. Есть три курса в формате “[Номер курса] [Код курса] [Название курса]”. Интервал между словами разный. Как разбить строку, разделенную регулярным выражением?

Их можно разбить двумя способами:

- 1) используя метод `re.split`;
- 2) вызвав метод `split` для объекта `regex`.

```
import re
text = """100 ИНФ Информатика 213 МАТ Математика 156
АНГ Английский"""
x=re.split('\s+', text)
print(x)
```

```
['100', 'ИНФ', 'Информатика', '213', 'МАТ', 'Математика', '156',
'АНГ', 'Английский']
```

или

```
import re
regex = re.compile('\s+')
text = """100 ИНФ Информатика 213 МАТ Математика 156
АНГ Английский"""
x=regex.split(text)
print(x)
```

```
['100', 'ИНФ', 'Информатика', '213', 'МАТ', 'Математика', '156',
'АНГ', 'Английский']
```

Пример. Найти все номера в тексте

```
import re
regex = re.compile('\s+')
text = """100 ИНФ Информатика 213 МАТ Математика 156
АНГ Английский"""
regex_num = re.compile('\d+')
x=regex_num.findall(text)
print(x)

['100', '213', '156']
```

В приведенном выше коде специальный символ \ d является регулярным выражением, которое соответствует любой цифре. Добавление к нему символа + означает наличие по крайней мере 1 числа.

Подобно +, есть символ *, для которого требуется 0 или более чисел. Это делает наличие цифры не обязательным, чтобы получилось совпадение.

Пример. Требуется извлечь номер курса, код и имя как отдельные элементы.

```
import re
text = """100 ИНФ Информатика
213 МАТ Математика
156 АНГ Английский"""
# извлечь все номера курсов
a=re.findall('[0-9]+', text)
# извлечь все коды курсов (для латиницы [A-Z])
b=re.findall('[A-ЯЁ]{3}', text)
# извлечь все названия курсов
c=re.findall('[a-яA-ЯёЁ]{4,}', text)
print(a)
print(b)
print(c)
['100', '213', '156']
['ИНФ', 'МАТ', 'АНГ']
['Информатика', 'Математика', 'Английский']
```

Для номера курса, шаблон [0-9]+ указывает на соответствие всем числам от 0 до 9. Добавление символа + в конце заставляет найти по крайней мере 1 соответствие цифрам 0-9. Если вы уверены, что номер курса, будет иметь ровно 3 цифры, шаблон мог бы быть [0-9] {3}.

Для кода курса, как вы могли догадаться, [А-ЯЁ]{3} будет совпадать с 3 большими буквами алфавита А-Я подряд (буква “ё” не включена в общий диапазон букв).

Для названий курса, [а-яА-ЯёЁ]{4,} будем искать а-я верхнего и нижнего регистра, предполагая, что имена всех курсов будут иметь как минимум 4 символа.

Пример. Замена совпадений на определенную строку

```
import re
text = "Пойди найди то не знаю что"
r = re.sub("\s", "-", text)
print(r)
```

Практика

Задача № 1. Извлеките никнейм пользователя, имя домена и суффикс из данных email адресов.

```
emails = ""zuck26@facebook.com
page33@google.com
jeff42@amazon.com""
```

```
# требуемый вывод
[('zuck26', 'facebook', 'com'), ('page33', 'google', 'com'), ('jeff42', 'amazon', 'com')]
```

Задача № 2. Извлеките все слова, начинающиеся с ‘b’ или ‘B’ из данного текста.

```
text = ""Betty bought a bit of butter, But the butter was so bitter,
So she bought some better butter, To make the bitter butter better.""
```

```
# требуемый вывод
['Betty', 'bought', 'bit', 'butter', 'But', 'butter', 'bitter', 'bought', 'better', 'butter', 'bitter', 'butter', 'better']
```

Задачи для самостоятельной работы

1. Из заданной строки получить список слов, у которых первая буква гласная, а вторая – согласная.

2. Из заданной строки получить список слов, у которых вторая буква согласная, а третья – гласная.

3. Из заданной строки получить список слов, у которых последняя буква гласная, а предпоследняя – согласная.

4. Из заданной строки получить список слов, у которых третья с конца буква согласная, а предпоследняя – гласная.

5. Напишите программу, которая посчитает количество смайликов в заданном тексте.

Смайликом будем считать последовательность символов, удовлетворяющая условиям:

- первым символом является либо «;» (точка с запятой) либо «:» (двоеточие) ровно один раз;

- далее может идти символ «-» (минус) сколько угодно раз (в том числе символ минус может идти ноль раз);

- в конце обязательно идет некоторое количество (не меньше одной) одинаковых скобок из следующего набора: «(», «)», «[», «]»;

- внутри смайлика не может встречаться никаких других символов.

Например, эта последовательность является смайликом: «;-----[[[[[[[».

Эти последовательности смайликами не являются: «]», «;--», «:», «(».

6. В заданном тексте заменить последовательность символов «a...ab...bc...c» (букв а и с в последовательности больше 0, букв b – больше единицы) на последовательность «qqq».

7. В заданной строке все пары символов, первый из которых – малая латинская буква, а второй – большая латинская буква, выделить знаками « _ ? » с обеих сторон.

8. Написать регулярное выражение, определяющее, является ли заданная строка правильным MAC-адресом. Пример правильного выражения: aE:dC:cA:56:76:54. Пример неправильного выражения: 01:23:45:67:89:Az.

9. Написать регулярное выражение, определяющее, является ли данная строка шестнадцатеричным идентификатором цвета в HTML, где #FFFFFF – белый цвет, #000000 – черный, #FF0000 – красный и т. д.

10. Примеры правильных выражений: #FFFFFF, #FF3421, #00ff00. Примеры неправильных выражений: 232323, f#fddee, #fd2.

11. Написать регулярное выражение, определяющее, является ли данная строка валидным e-mail адресом. Примеры правильных выражений: user@example.com, root@localhost. Примеры неправильных выражений: bug@@@com.ru, @val.ru, Just Text2.

12. Составить регулярное выражение, определяющее, является ли заданная строка IP-адресом, записанным в десятичном виде. Примеры правильных выражений: 127.0.0.1, 255.255.255.0. Примеры неправильных выражений: 1300.6.7.8, abc.def.gha.bcd.

13. Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов, где символом может быть английская буква, цифра или знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру.

14. Примеры правильных выражений: C00l_Pass, SupperPas1. Примеры неправильных выражений: Cool_pass, C00l.

15. Проверить, является ли заданная строка шестизначным числом, записанным в десятичной системе счисления без нулей в старших разрядах. Примеры правильных выражений: 123456, 234567. Примеры неправильных выражений: 1234567, 12345.

16. Есть текст со списками цен. Извлечь из него цены в USD, RUR, EU. Примеры правильных выражений: 23.78 USD. Примеры неправильных выражений: 22 UDD, 0.002 USD.

17. Проверить, существуют ли в тексте цифры, за которыми стоит «+». Пример правильного выражения: $(3 + 5) - 9 \times 4$. Пример неправильного выражения: $2 * 9 - 6 \times 5$.

18. Время имеет формат «часы : минуты». И часы, и минуты состоят из двух цифр, пример: 09:00. Напишите регулярное выражение для поиска времени в заданной строке.

19. Арифметическое выражение состоит из двух чисел и операции между ними, например: $1 + 2$; $1.2 * 3.4$; $-3 / -6$; $-2 - 2$. Список операций: «+», «-», «*» и «/». Также могут присутствовать пробелы вокруг оператора и чисел. Напишите регулярное выражение, которое найдёт все арифметические выражения в заданной строке.

20. Написать регулярное выражение, определяющее, является ли данная строка датой в формате «dd/mm/yyyy», начиная с 1600 до 9999 года включительно.

21. Составить регулярное выражение, проверяющее, содержит ли введенная строка вещественное число.

22. В России применяются регистрационные знаки нескольких видов. Общего в них то, что они состоят из цифр и букв. Причём используются только 12 букв кириллицы, имеющие графические аналоги в латинском алфавите – А, В, Е, К, М, Н, О, Р, С, Т, У и Х.

У частных легковых автомобилей номера – это буква, три цифры, две буквы, затем две или три цифры с кодом региона. У такси – две буквы, три цифры, затем две или три цифры с кодом региона. Требуется определить, является ли последовательность букв корректным номером указанных двух типов, и если является, то каким.

Лабораторная работа № 12. СЛОВАРИ

Цель работы – познакомиться с основными конструкциями, используемыми для работы со словарями; приемами программной реализации на языке программирования Python; произвести отладку и тестирование полученных программ.

Задание:

1. Изучить краткие теоретические сведения.
2. Разработать алгоритмы для решения задач (блок-схемы).
3. Записать алгоритмы на языке программирования Python.
4. Произвести отладку и тестирование программы.
5. Оформить отчет по лабораторной работе.

Отчет о лабораторной работе должен содержать следующие сведения:

- 1) название и цель работы;
- 2) формулировку задачи, схему алгоритма, программный код и результаты решения задачи;
- 3) вывод по работе в целом.

Краткие теоретические сведения

Словари (dict) – это одна из наиболее часто используемых структур данных, позволяющая хранить объекты, для доступа к которым используется ключ.

Описание словаря имеет вид:

{ключ: значение, ключ: значение, ключ: значение, ...}

Ключом может являться любой неизменяемый тип данных (строки, целые числа, кортежи). Значением конкретного ключа может быть любой тип данных. Ключи в словарях всегда уникальны.

Чтобы представление о словаре стало более понятным, можно провести аналогию с обычным словарем, например, англо-русским.

На каждое английское слово в таком словаре есть русское слово-перевод:

cat – кошка, dog – собака, table – стол и т. д.

Если англо-русский словарь описать с помощью Python, то английские слова можно сделать **ключами**, а русские – их **значениями**:

```
{'cat': 'кошка', 'dog': 'собака', 'bird': 'птица', 'mouse': 'мышь'}
```

Способы задания словарей

1. Словарь можно задать путем перечисления ключей и значений, разделённых двоеточием, отделённых друг от друга запятыми и заключённых в фигурные скобки «{ }».

Примеры:

```
d1 = {"house": "дом", "car": "машина", "tree": "дерево", "road": "дорога"}
d2 = {True: 'Истина', False: 'Ложь'}
d3 = {'fruit': 'mango', 1: [4, 6, 8]} #ключи разных типов
d4 = {} #пустой словарь
d41=dict()
d5 = {1: {"house": "дом", "car": "машина", "tree": "дерево", "road": "дорога"},
      2: {True: 'Истина', False: 'Ложь'}} #словарь внутри словаря
print(d1)
print(d2)
print(d3)
print(d4)
print(d41)
print(d5) |
```

```
{'house': 'дом', 'car': 'машина', 'tree': 'дерево', 'road': 'дорога'}
{True: 'Истина', False: 'Ложь'}
{'fruit': 'mango', 1: [4, 6, 8]}
{}
{}
{1: {'house': 'дом', 'car': 'машина', 'tree': 'дерево', 'road': 'дорога'}, 2: {True: 'Истина', False: 'Ложь'}}
```

2. Словарь можно определить с помощью функции-конструктора **dict()**:

Примеры:

```
d1 = dict([(1, 'mango'), (2, 'rawraw')])
d2 = dict(house = "дом", car = "машина", tree = "дерево", road = "дорога")
print(type(d2))
print(d1)
print(d2)
```

```
<class 'dict'>
{1: 'mango', 2: 'rawraw'}
{'house': 'дом', 'car': 'машина', 'tree': 'дерево', 'road': 'дорога'}
```

Здесь ключи записываются уже **без кавычек** и после них ставится **знак равно** вместо двоеточий. В результате, получаем точно такой же словарь. И такой способ задания работает только, если в качестве ключей выступают строки. Например, с числами такой подход работать не будет:

```
d3 = dict(1 = "дом", 2 = "машина")
print(d3)
```

```
File "C:/Users/AMakhnenko/Desktop/z-e11_v2_5.py", line 1
    d3 = dict(1 = "дом", 2 = "машина")
              ^
SyntaxError: keyword can't be an expression
```

3. С помощью генераторов словарей.

Пример 1:

```
d = {a: a ** 2 for a in range(7)}
print(d)
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Пример 2:

```
Cities = ["Moscow", "Kiev", "Washington"]
States = ["Russia", "Ukraine", "USA"]
dic = {state: city for city, state in zip(Cities, States)}
print(dic)
```

```
{'Russia': 'Moscow', 'Ukraine': 'Kiev', 'USA': 'Washington'}
```

4. С помощью метода **fromkeys**:

Примеры:

```
d = dict.fromkeys(['a', 'b'])
print(d)
d = dict.fromkeys(['a', 'b'], 100)
print(d)
```

```
{'a': None, 'b': None}
{'a': 100, 'b': 100}
```

Доступ к элементам из словаря

Для доступа к значениям в словаре используются ключи. Ключ можно использовать либо в квадратных скобках «[]», либо с помощью функции `get()`.

Разница при использовании `get()` заключается в том, что он возвращает `None` вместо `KeyError`, если ключ не найден.

Пример 1:

```
my_dict = {'name': 'Jack', 'age': 26}
print(my_dict['name'])

Jack
my_dict = {'name': 'Jack', 'age': 26}
print(my_dict['nam'])

File "C:/Users/AMakhnenko/Desktop/z-e11_v2_5.py", line 2, in <module>
    print(my_dict['nam'])

KeyError: 'nam'
```

Пример 2:

```
my_dict = {'name': 'Jack', 'age': 26}
print(my_dict.get('age'))
print(my_dict.get('ag'))

26
None
```

Пример 3:

Если в словаре записать два одинаковых ключа, то ключ будет ассоциирован с последним указанным значением.

```
d = {"house": "дом", "house": "дом 2", "car": "машина"}
print(d["house"])

дом 2
```

Изменение и добавление элементов в словарь

Словарь – изменяемый тип данных. Мы можем добавить новые элементы или изменить значение существующих элементов, используя оператор присваивания.

Если ключ уже присутствует, значение обновляется, иначе в словарь добавляется новая пара:

ключ: значение.

Пример:

```
my_dict = {'name': 'Иван', 'age': 26}
# обновляем значение
my_dict['age'] = 27
print(my_dict)
# добавляем элемент
my_dict['address'] = 'Братск'
print(my_dict)

{'name': 'Иван', 'age': 27}
{'name': 'Иван', 'age': 27, 'address': 'Братск'}
```

Удаление элементов из словаря

Удаление элементов в словаре осуществляется следующими способами:

- используя метод **pop()**. Удаляет элемент с указанным ключом;
- используя метод **popitem()**. Удаление и возврата произвольного элемента (ключ, значение) из словаря;
- используя метод **clear()**. Удаление всех элементов словаря;
- используя ключевое слово **del**. Удаление отдельных элементов или всего словаря.

Примеры:

```
# создаем словарь
squares = {1:1, 2:4, 3:9, 4:16, 5:25}
# удалить определенный элемент словаря
print(squares.pop(4))
print(squares)
print()
# удалить случайный элемент из словаря
print(squares.popitem())
print(squares)
print()
# удалить определенный элемент словаря
del squares[2]
print(squares)
print()
# удалить все из словаря
squares.clear()
print(squares)
print()
# удалить словарь
del squares

16
{1: 1, 2: 4, 3: 9, 5: 25}

(5, 25)
{1: 1, 2: 4, 3: 9}

{1: 1, 3: 9}

{}
```

Операции со словарем

1. Проверка на наличие элемента в словаре – оператор **in**.

Мы можем проверить, находится ли ключ в словаре или нет. Обратите внимание, что проверка на наличие элемента предназначена только для ключей, а не для значений.

Пример:

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
print(1 in squares)
print(2 not in squares)
print(49 in squares)

True
True
False
```

2. Перебор элементов словаря – цикл **for**.

Пример 1. Элементы словаря перебираются в цикле **for** также, как элементы других сложных объектов. Однако "по умолчанию" извлекаются только ключи:

```
nums = {1: 'ИСИТ', 2: 'ИПО', 3: 'УТС', 4: 'МТС'}
for i in nums:
    print(i)

1
2
3
4
```

Пример 2. Но по ключам всегда можно получить значения:

```
nums = {1: 'ИСИТ', 2: 'ИПО', 3: 'УТС', 4: 'МТС'}
for i in nums:
    print(nums[i])

ИСИТ
ИПО
УТС
МТС
```

Пример 3. У словаря есть метод **items()**, который создает особую структуру, состоящую из кортежей. Каждый кортеж включает ключ и значение. В цикле **for** можно распаковывать кортежи, таким образом сразу извлекая как ключ, так и его значение:

```

nums = {1: 'ИСИТ', 2: 'ИПО', 3: 'УТС',4:'МТС'}
n = nums.items()
print(n)
for key, value in nums.items():
    print(key, 'is', value)

dict_items([(1, 'ИСИТ'), (2, 'ИПО'), (3, 'УТС'), (4, 'МТС')])
1 is ИСИТ
2 is ИПО
3 is УТС
4 is МТС

```

Пример 4. Методы словаря **keys()** и **values()** позволяют получить отдельно перечни ключей и значений. Так что если, например, надо перебрать только значения или только ключи, лучше воспользоваться одним из этих методов:

```

nums = {1: 'ИСИТ', 2: 'ИПО', 3: 'УТС',4:'МТС'}
v_nums = []
for v in nums.values():
    v_nums.append(v)
print(v_nums)
k_nums = []
for k in nums.keys():
    k_nums.append(k)
print(k_nums)

['ИСИТ', 'ИПО', 'УТС', 'МТС']
[1, 2, 3, 4]

```

Встроенные функции

Функция	Значение
len()	Возвращает число элементов словаря, т.е. количество пар ключ:значение
all()	Возвращает True, если все ключи словаря истинны (или если словарь пуст)
any()	Возвращает True, если любой ключ словаря равен true. Если словарь пуст, вернет False
sorted()	Возвращает новый отсортированный список ключей в словаре
str()	Преобразует словарь в строку
eval()	Конвертация строки в словарь

Примеры:

```
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
print(len(squares))
print(sorted(squares))
print(str(squares))
print(all(squares))
print(any(squares))
a = '{1: "ИСИТ", 2: "ИПО", 3: "УТС",4:"МТС"}'
b = eval(a)
print(b)

5
[1, 3, 5, 7, 9]
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
True
True
{1: 'ИСИТ', 2: 'ИПО', 3: 'УТС', 4: 'МТС'}
```

Методы для работы со словарями

Метод	Назначение
update	Объединение содержимого двух словарей в один
fromkeys	Создает новый словарь
items	Возвращает пары (ключи и значения)
keys	Возвращает ключи
values	Возвращает значения
copy	Копирует содержимое в другой словарь
clear	Полная очистка всех элементов словаря
pop	Удаляет из словаря элемент по указанному ключу

Примеры решения задач

Задача 1

1. Создать произвольный словарь.
2. Добавить новый элемент с ключом типа str и значением типа int.
3. Добавить новый элемент с ключом типа кортеж (tuple) и значением типа список (list).
4. Получить элемент по ключу.
5. Удалить элемент по ключу.
6. Получить список ключей словаря.


```

#1
dict = {1: 'Петя', 2: 'Маша', 3: 'Саша'}
print('#1')
print(dict)
#2
dict['Ключ'] = 12345
print('#2')
print(dict)
#3
dict[('Это', 'будет', 'кортеж')] = [11, 22, 'Значение_List', 33, {1, 2, 3}]
print('#3')
print(dict)
#4
print('#4')
item_by_key_v1 = dict['Ключ']
print(item_by_key_v1)
item_by_key_v2 = dict.get('Ключ', 'No item')
print(item_by_key_v2)
#5
item_deleted = dict.pop(2, 'No item')
print('#5')
print(dict)
#6
keys = dict.keys()
print('#6')
print(keys)

#1
{1: 'Петя', 2: 'Маша', 3: 'Саша'}
#2
{1: 'Петя', 2: 'Маша', 3: 'Саша', 'Ключ': 12345}
#3
{1: 'Петя', 2: 'Маша', 3: 'Саша', 'Ключ': 12345, ('Это', 'будет', 'кортеж'): [11, 22, 'Значение_List', 33, {1, 2, 3}]}
#4
12345
12345
#5
{1: 'Петя', 3: 'Саша', 'Ключ': 12345, ('Это', 'будет', 'кортеж'): [11, 22, 'Значение_List', 33, {1, 2, 3}]}
#6
dict_keys([1, 3, 'Ключ', ('Это', 'будет', 'кортеж')])

```

Задача 2. Написать функцию `month_to_season()`, которая принимает 1 аргумент – номер месяца – и возвращает название сезона, к которому относится этот месяц.

Например, передаем 2, на выходе получаем 'Зима'.

```

def month_to_season(month):
    season_ranges = {
        (12, 1, 2): 'Зима',
        (3, 4, 5): 'Весна',
        (6, 7, 8): 'Лето',
        (9, 10, 11): 'Осень'
    }

```

```

season = None # переменная для возвращаемого значения функции
for key, value in season_ranges.items():
    if month in key:
        season = value
        break
return season
print(month_to_season(1))
print(month_to_season(5))
print(month_to_season(8))
print(month_to_season(9))
print(month_to_season(12))
print(month_to_season(24))

Зима
Весна
Лето
Осень
Зима
None

```

Задача 3. Создать словарь *Capitals*, где индексом является название страны, а значением – название столицы этой страны. Определять по строке с названием страны ее столицу.

```

# Создаем пустой словарь Capitals
Capitals = {}
# Заполняем его несколькими значениями
Capitals['Россия'] = 'Москва'
Capitals['Украина'] = 'Киев'
Capitals['США'] = 'Вашингтон'
Capitals['Англия'] = 'Лондон'
Capitals['Италия'] = 'Рим'
Capitals['Израиль'] = 'Иерусалим'
# Считаем название страны
print('В какой стране вы живете?')
country = input()
# Проверяем, есть ли такая страна в словаре Capitals
if country in Capitals:
    # Если есть - выводим ее столицу
    print('Столица вашей страны', Capitals[country])
else:
    # Запросим название столицы и добавим его в словарь
    print('Как называется столица вашей страны?')
    city = input()
    Capitals[country] = city
print(Capitals)

```

В какой стране вы живете?

Россия

Столица вашей страны Москва

```
{'Россия': 'Москва', 'Украина': 'Киев', 'США': 'Вашингтон', 'Англия': 'Лондон', 'Италия': 'Рим',
'Израиль': 'Иерусалим'}
```

Задача 4. Дан список стран и городов каждой страны. Затем даны названия городов. Для каждого города укажите, в какой стране он находится.

```
d=dict()
lc=[]
lt=[]
k=int(input()) #количество стран
for i in range(k):
    s=input('Введите строку: ') #ввод названия страны и городов
    l=s.split()
    print(l)
    lc.insert(len(lc),l[0]) #список стран
    print(lc)
    lt.insert(len(lt),l[1:len(l)]) #список городов
    print(lt)
d=dict(zip(lc,lt)) #словарь
print(d)
k=int(input()) #количество городов
for i in range(k):
    city=input('Введите строку: ') #ввод названия города
    for key, value in d.items(): #поиск страны с данным городом
        if city in value:
            print(key)
```

2

Введите строку: Россия Москва Братск

['Россия', 'Москва', 'Братск']

['Россия']

[['Москва', 'Братск']]

Введите строку: Украина Одесса Очаков

['Украина', 'Одесса', 'Очаков']

['Россия', 'Украина']

[['Москва', 'Братск'], ['Одесса', 'Очаков']]

{'Россия': ['Москва', 'Братск'], 'Украина': ['Одесса', 'Очаков']}

3

Введите строку: Братск

Россия

Введите строку: Москва

Россия

Введите строку: Очаков

Украина

Задачи для самостоятельной работы

1. Опишите, используя словарь, телефонную книгу. Составьте программу, выдающую список абонентов, имеющих телефонный номер, начинающийся на 33.

2. Опишите, используя словарь, каталог книг в библиотеке. Составьте программу, выдающую список книг В. Пикуля, хранящихся в библиотеке.

3. Опишите, используя словарь, таблицу дат и событий русской истории. Составьте программу, выдающую список событий XIX века.

4. Опишите, используя словарь, школьную нагрузку (фамилия преподавателя, класс, часы). Составьте программу, определяющую нагрузку каждого преподавателя. Определить, у какого преподавателя самая большая нагрузка и у какого самая низкая.

5. Опишите, используя словарь, таблицу соревнований (название команды, количество набранных очков). Выбрать команду, занявшую первое место. Упорядочить список команд, в зависимости от занятого места.

6. При сдаче норм ГТО были получены результаты забега на 100 метров и прыжков в длину. Задайте нормы ГТО по этим видам, определите списки учеников, не выполнивших нормативы, количество учеников, сдавших нормативы, а также список 3 лучших.

7. Опишите, используя словарь, товар (наименование товара, старая цена, новая цена). Составьте программу, определяющую, на какие товары повысятся цены и на сколько процентов.

8. В анкетных данных обозначены фамилия, пол, рост. Определите средний рост женщин, фамилию самого высокого мужчины, есть ли в группе хотя бы два человека одного роста.

9. Опишите, используя словарь, записную книжку (фамилия, номер телефона). Составьте программу, определяющую: 1) есть ли в записной книжке сведения о знакомом с фамилией на букву «Ф», если есть – напечатайте его фамилию и телефон; 2) есть ли в записной книжке сведения о знакомом с телефоном 47-67-35, если есть – напечатайте его фамилию.

10. Вам дан словарь, состоящий из слов, расположенных парами. Каждое слово является синонимом к парному ему слову. Все слова в словаре различны. Для последнего слова из словаря определите его синоним.

11. Дан список стран и городов каждой страны. Затем даны названия городов. Для каждого города укажите, в какой стране он находится.

12. В сводке об экспортируемых товарах указывается: наименование товара, страна, импортирующая товар, объем поставляемой партии в штуках. Напечатайте списки стран, в которые экспортируется данный товар, и общий объем его экспорта.

13. Хранятся сведения о лесе: вид дерева, общая численность, численность здоровых деревьев. Составьте программу вычисления:

- 1) суммарного числа деревьев на контрольном участке;
- 2) суммарного числа здоровых деревьев;
- 3) относительную численность (%) больных деревьев;
- 4) относительную численность (%) различных видов, в том числе больных (%) для каждого вида.

14. При поступлении на музыкально-педагогический факультет на абитуриентов собирают информацию: фамилия, музыкальный инструмент. Для поступления необходимо сдать экзамен по специальности. Составьте списки для данного экзамена, в зависимости от специальности.

15. Опишите, используя словарь, школьный класс (фамилия и инициалы, дата рождения, месяц рождения, год рождения). Напечатайте список учеников, рожденных в мае.

16. Опишите, используя словарь, записную книжку. Напечатайте список друзей, кому в этом году исполняется 25 лет (фамилия и инициалы, год рождения, дата рождения, месяц рождения).

17. Опишите, используя словарь, школьный класс (фамилия и инициалы, дата рождения, месяц рождения, год рождения). Вычислите день рождения класса (среднее арифметическое дат, месяцев, годов).

18. Опишите, используя словарь, данные на учеников (фамилия, улица, дом, квартира). Составьте программу, определяющую, сколько учеников живет на улице Гоголя, списки учеников, живущих в доме номер 45.

19. Опишите, используя словарь, выборы (фамилия кандидата и количество набранных голосов). Всего избирателей 2000. Определить, кто из кандидатов прошел, или необходимо проводить повторные выборы (должно быть набрано не менее 1/3 голосов от общего количества).

20. Выберите самую высокую вершину из заданного словаря.

21. Опишите, используя словарь, анкету школьника (фамилия, возраст). Определите возрастные группы в классе и напечатайте их списки.

22. Опишите, используя словарь, оценки за год. Посчитайте процент и качество успеваемости в классе за год, составьте списки неуспевающих и отличников.

23. О поступивших в вуз студентах собрана информация: фамилия, нуждается ли в общежитии, стаж работы (если есть), что окончил, какой язык изучал. Определите:

- а) сколько человек нуждаются в общежитии;
- б) списки студентов, имеющих стаж работы более 2 лет;
- в) списки окончивших техникум;
- г) списки языковых групп.

Лабораторная работа № 13. ФАЙЛЫ

Цель работы – познакомиться с основными конструкциями, используемыми для работы с файлами; приемами программной реализации на языке программирования Python; произвести отладку и тестирование полученных программ.

Задание:

1. Изучить краткие теоретические сведения.
2. Разработать алгоритмы для решения задач (блок-схемы).
3. Записать алгоритмы на языке программирования Python.
4. Произвести отладку и тестирование программы.
5. Оформить отчет по лабораторной работе.

Отчет о лабораторной работе должен содержать следующие сведения:

- 1) название и цель работы;
- 2) формулировку задачи, схему алгоритма, программный код и результаты решения задачи;
- 3) вывод по работе в целом.

Краткие теоретические сведения

Python поддерживает множество различных типов файлов, но условно их можно разделить на два вида: текстовые и бинарные. Текстовые файлы – это к примеру файлы с расширением `cvs`, `txt`, `html`, в общем любые файлы, которые сохраняют информацию в текстовом виде. Бинарные файлы – это изображения, аудио и видеофайлы и т. д. В зависимости от типа файла работа с ним может немного отличаться.

При работе с файлами необходимо соблюдать некоторую последовательность операций:

- 1) открытие файла с помощью метода **`open()`**;
- 2) чтение файла с помощью метода **`read()`** или запись в файл посредством метода **`write()`**;
- 3) закрытие файла методом **`close()`**.

Открытие и закрытие файла

Чтобы начать работу с файлом, его надо открыть с помощью функции **open()**, которая имеет следующее формальное определение:

```
open(file, mode)
```

Первый параметр функции представляет путь к файлу. Путь файла может быть абсолютным, то есть начинаться с буквы диска, например, *C://somedir/somefile.txt*. Либо можно быть относительным, например, *somedir/somefile.txt* – в этом случае поиск файла будет идти относительно расположения запущенного скрипта Python.

Второй передаваемый аргумент – **mode** устанавливает режим открытия файла в зависимости от того, что мы собираемся с ним делать.

Существует 4 общих режима:

- ♦ **r** (Read). Файл открывается для чтения. Если файл не найден, то генерируется исключение `FileNotFoundError`

- ♦ **w** (Write). Файл открывается для записи. Если файл отсутствует, то он создается. Если подобный файл уже есть, то он создается заново, и соответственно старые данные в нем стираются.

- ♦ **a** (Append). Файл открывается для дозаписи. Если файл отсутствует, то он создается. Если подобный файл уже есть, то данные записываются в его конец.

- ♦ **b** (Binary). Используется для работы с бинарными файлами. Применяется вместе с другими режимами – **w** или **r**.

После завершения работы с файлом его обязательно нужно закрыть методом `close()`. Данный метод освободит все связанные с файлом используемые ресурсы.

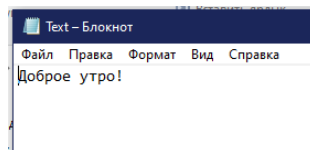
Например, откроем для записи текстовый файл "Text.txt":

```
1 myfile = open("Text.txt", "w")  
2  
3 myfile.close()
```

При открытии файла или в процессе работы с ним мы можем столкнуться с различными исключениями, например, к нему нет доступа и т.д. В этом случае программа выпадет в ошибку, а ее выполнение не дойдет до вызова метода `close`, и соответственно файл не будет закрыт.

В этом случае мы можем обрабатывать исключения:

```
1 try:
2     a = open("Text.txt", "w")
3     try:
4         a.write("Доброе утро!")
5     except Exception as text:
6         print(text)
7     finally:
8         a.close()
9 except Exception as text1:
10    print(text1)
```



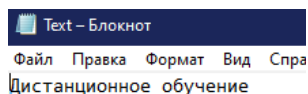
В данном случае вся работа с файлом идет во вложенном блоке try. И если вдруг возникнет какое-либо исключение, то в любом случае в блоке finally файл будет закрыт.

Однако есть и более удобная конструкция – конструкция **with**:

with open(file, mode) as file_obj:
инструкции

Эта конструкция определяет для открытого файла переменную file_obj и выполняет набор инструкций. После их выполнения файл автоматически закрывается. Даже если при выполнении инструкций в блоке with возникнут какие-либо исключения, то файл все равно закрывается.

```
1 with open("Text.txt", "w") as Text:
2     Text.write("Дистнционное обучение")
```



Запись в текстовый файл

Чтобы открыть текстовый файл на запись, необходимо применить режим **w** (перезапись) или **a** (дозапись). Затем для записи применяется метод **write(str)**, в который передается записываемая строка. Стоит отметить, что записывается именно строка, поэтому, если нужно записать числа, данные других типов, то их предварительно нужно конвертировать в строку.

Запишем некоторую информацию в файл "Text.txt":

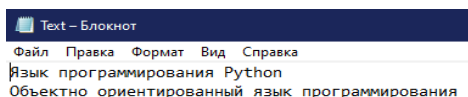
```
1 with open("Text.txt", "w") as a:
2     a.write("Язык программирования Python")
3
```

Если мы откроем папку, в которой находится текущий скрипт Python, то увидим там файл Text.txt. Этот файл можно открыть в любом текстовом редакторе и при желании изменить.

Теперь дозапишем в этот файл еще одну строку:

```
1 with open("Text.txt", "a") as a:
2     a.write("\nОбъектно ориентированный язык программирования")
```

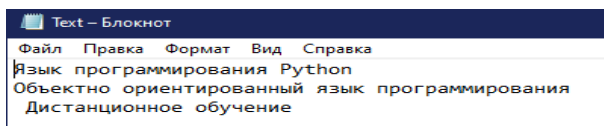
Дозапись выглядит как добавление строки к последнему символу в файле, поэтому, если необходимо сделать запись с новой строки, то можно использовать эскейп-последовательность `"\n"`. В итоге файл Text.txt будет иметь следующее содержимое:



Еще один способ записи в файл представляет стандартный метод `print()`, который применяется для вывода данных на консоль:

```
1 with open("Text.txt", "a") as a:
2     print("Дистанционное обучение", file=a)
```

Для вывода данных в файл в метод `print` в качестве второго параметра передается название файла через параметр `file`. А первый параметр представляет записываемую в файл строку.



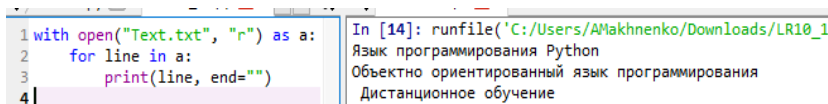
Чтение файла

Для чтения файла он открывается с режимом `r` (Read), и затем мы можем считать его содержимое различными методами:

- ♦ **readline()**: считывает одну строку из файла
- ♦ **read()**: считывает все содержимое файла в одну строку
- ♦ **readlines()**: считывает все строки файла в список

Например, считаем выше записанный файл построчно:

```
1 with open("Text.txt", "r") as a:
2     for line in a:
3         print(line, end="")
4
```



Несмотря на то, что мы явно не применяем метод **readline()** для чтения каждой строки, но в при переборе файла этот метод автоматически вызывается для получения каждой новой строки. Поэтому в цикле вручную нет смысла вызывать метод **readline**. И поскольку строки разделяются символом перевода строки `"\n"`, то чтобы исключить излишнего переноса на другую строку в функцию **print** передается значение `end=""`.

Теперь явным образом вызовем метод **readline()** для чтения отдельных строк:

```
1
2 with open("Text.txt", "r") as a:
3     stroka1 = a.readline()
4     print(stroka1, end="")
5     stroka2 = a.readline()
6     print(stroka2)
7     stroka3=a.readline()
8     print(stroka3)
```

In [20]: runfile('C:/Users/AMakhnenko/Downloads/1
Язык программирования Python
Объектно ориентированный язык программирования
Дистанционное обучение

Метод **readline** можно использовать для построчного считывания файла в цикле **while**:

```
2 with open("Text.txt", "r") as a:
3     stroka = a.readline()
4     while stroka:
5         print(stroka, end="")
6         stroka = a.readline()
7
```

Язык программирования Python
Объектно ориентированный язык программирования
Дистанционное обучение
In [23]:

Если файл небольшой, то его можно разом считать с помощью метода **read()**:

```
2 with open("Text.txt", "r") as a:
3     content = a.read()
4     print(content)
```

Язык программирования Python
Объектно ориентированный язык программирования
Дистанционное обучение

И также применим метод **readlines()** для считывания всего файла в список строк:

```
2 with open("Text.txt", "r") as a:
3     contents = a.readlines()
4     stroka1 = contents[0]
5     stroka2 = contents[1]
6     print(stroka1, end="")
7     print(stroka2)
```

Язык программирования Python
Объектно ориентированный язык программирования
In [27]:

При чтении файла мы можем столкнуться с тем, что его кодировка не совпадает с ASCII. В этом случае мы явным образом можем указать кодировку с помощью параметра **encoding**:

```
2 filename = "Text.txt"
3 with open(filename, encoding="utf8") as a:
4     text = a.read()
```

Пример: написать скрипт, в котором будет записываться введенный пользователем массив строк и считываться обратно из файла на консоль:

```
2 # имя файла
3 name = "messages.txt"
4 # определяем пустой список
5 spisok = list()
6
7 for i in range(4):
8     stroka = input("Введите строку " + str(i+1) + ": ")
9     spisok.append(stroka + "\n")
10
11 # запись списка в файл
12 with open(name, "a") as a:
13     for stroka in spisok:
14         a.write(stroka)
15
16 # считываем сообщения из файла
17 print("Считанные сообщения")
18 with open(name, "r") as a:
19     for stroka in a:
20         print(stroka, end="")
21
```

Введите строку 1: Уважаемый Степан Иванович!

Введите строку 2: Вам необходимо появиться в деканате 16 ноября 2020 года, в 13-00!

Введите строку 3: Надеемся на сотрудничество!

Введите строку 4: До встречи!

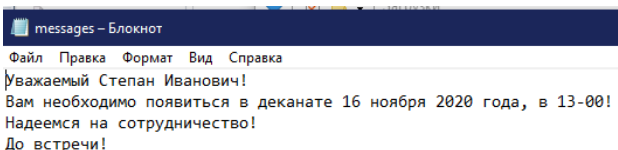
Считанные сообщения

Уважаемый Степан Иванович!

Вам необходимо появиться в деканате 16 ноября 2020 года, в 13-00!

Надеемся на сотрудничество!

До встречи!



messages - Блокнот

Файл Правка Формат Вид Справка

Уважаемый Степан Иванович!

Вам необходимо появиться в деканате 16 ноября 2020 года, в 13-00!

Надеемся на сотрудничество!

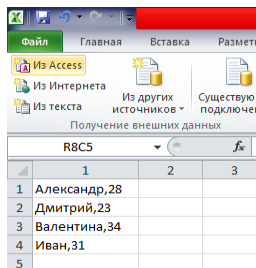
До встречи!

Файлы CSV

Одним из распространенных файловых форматов, которые хранят в удобном виде информацию, является формат **csv**. Каждая строка в файле **csv** представляет отдельную запись или строку, которая состоит из отдельных столбцов, разделенных запятыми. Соответственно поэтому формат и называется Comma Separated Values. Но хотя формат **csv** – это формат текстовых файлов, Python для упрощения работы с ним предоставляет специальный встроенный модуль **csv**.

Рассмотрим работу модуля на примере:

```
1
2 import csv
3
4 name = "Users.csv"
5
6 users = [
7     ["Александр", 28],
8     ["Дмитрий", 23],
9     ["Валентина", 34]
10 ]
11
12 with open(name, "w", newline="") as a:
13     c = csv.writer(a)
14     c.writerows(users)
15
16
17 with open(name, "a", newline="") as a:
18     user = ["Иван", 31]
19     c = csv.writer(a)
20     c.writerow(user)
21
```



	1	2	3
1	Александр,28		
2	Дмитрий,23		
3	Валентина,34		
4	Иван,31		
5			

В файл записывается двухмерный список – фактически таблица, где каждая строка представляет одного пользователя. А каждый пользователь содержит два поля – имя и возраст. То есть фактически таблица из трех строк и двух столбцов.

При открытии файла на запись в качестве третьего параметра указывается значение `newline=""` – пустая строка позволяет корректно считывать строки из файла вне зависимости от операционной системы.

Для записи нам надо получить объект **writer**, который возвращается функцией `csv.writer(file)`. В эту функцию передается открытый файл. А собственно запись производится с помощью метода `writer.writerows(users)`. Этот метод принимает набор строк. В нашем случае это двухмерный список.

Если необходимо добавить одну запись, которая представляет собой одномерный список, например, `["Иван", 31]`, то в этом случае можно вызвать метод **writer.writerow(user)**.

Для чтения из файла нам наоборот нужно создать объект **reader**:

```
2 import csv
3
4 name = "Users.csv"
5
6 with open(name, "r", newline="") as a:
7     reader = csv.reader(a)
8     for row in reader:
9         print(row[0], " - ", row[1])
10
11
```

```
Александр - 28
Дмитрий   - 23
Валентина - 34
Иван      - 31
```

Работа со словарями

В примере выше каждая запись или строка представляла собой отдельный список, например, ["Иван", 31]. Но кроме того, модуль csv имеет специальные дополнительные возможности для работы со словарями. В частности, функция **csv.DictWriter()** возвращает объект writer, который позволяет записывать в файл. А функция **csv.DictReader()** возвращает объект reader для чтения из файла.

Например:

```
2 import csv
3
4 name = "Users.csv"
5
6 users = [
7     {"Имя": "Александр", "Возраст": 28},
8     {"Имя": "Дмитрий", "Возраст": 23},
9     {"Имя": "Валентина", "Возраст": 34}
10 ]
11
12 with open(name, "w", newline="") as a:
13     columns = ["Имя", "Возраст"]
14     writer = csv.DictWriter(a, fieldnames=columns)
15     writer.writeheader()
16
17     # запись нескольких строк
18     writer.writerows(users)
19
20     user = {"Имя": "Иван", "Возраст": 41}
21     # запись одной строки
22     writer.writerow(user)
23
24 with open(name, "r", newline="") as file:
25     reader = csv.DictReader(file)
26     for row in reader:
27         print(row["Имя"], "-", row["Возраст"])
```

Александр	-	28
Дмитрий	-	23
Валентина	-	34
Иван	-	41

Запись строк также производится с помощью методов **writerow()** и **writerows()**. Но теперь каждая строка представляет собой отдельный словарь, и кроме того, производится запись и заголовков столбцов с помощью метода **writeheader()**, а в метод **csv.DictWriter** в качестве второго параметра передается набор столбцов.

При чтении строк, используя названия столбцов, мы можем обратиться к отдельным значениям внутри строки: **row["name"]**.

Бинарные файлы

Бинарные файлы в отличие от текстовых хранят информацию в виде набора байт. Для работы с ними в Python необходим встроенный модуль **pickle**. Этот модуль предоставляет два метода:

- ♦ **dump(obj, file)**: записывает объект obj в бинарный файл file;
- ♦ **load(file)**: считывает данные из бинарного файла в объект.

При открытии бинарного файла на чтение или запись также надо учитывать, что нам нужно применять режим "b" в дополнение к режиму записи ("w") или чтения ("r"). Допустим, надо сохранить два объекта:

```
1 import pickle
2
3
4 FILENAME = "user.dat"
5
6 name = "Демид"
7 age = 19
8
9 with open(FILENAME, "wb") as file:
10     pickle.dump(name, file)
11     pickle.dump(age, file)
12
13 with open(FILENAME, "rb") as file:
14     name = pickle.load(file)
15     age = pickle.load(file)
16     print("Имя:", name, "\tВозраст:", age)
```

Имя: Демид Возраст: 19

user.dat	15.11.2020 1:13	Файл "DAT"	1 КБ
----------	-----------------	------------	------

С помощью функции **dump** последовательно записываются два объекта. Поэтому при чтении файла также последовательно посредством функции **load** мы можем считать эти объекты.

Подобным образом мы можем сохранять и извлекать из файла наборы объектов:

```
1 import pickle
2
3
4 FILENAME = "users.dat"
5
6 users = [
7     ["Александр", 28, True],
8     ["Дмитрий", 23, False],
9     ["Валентина", 34, False]
10 ]
11
12 with open(FILENAME, "wb") as file:
13     pickle.dump(users, file)
14
15
16 with open(FILENAME, "rb") as file:
17     users_from_file = pickle.load(file)
18     for user in users_from_file:
19         print("Имя:", user[0], "\tВозраст:", user[1], "\tЖенат(замужем):", user[2])
```

```
Имя: Александр  Возраст: 28      Женат(замужем): True
Имя: Дмитрий   Возраст: 23      Женат(замужем): False
Имя: Валентина Возраст: 34      Женат(замужем): False
```

Модуль *shelve*

Для работы с бинарными файлами в Python может применяться еще один модуль — **shelve**. Он сохраняет объекты в файл с определенным ключом. Затем по этому ключу может извлечь ранее сохраненный объект из файла. Процесс работы с данными через модуль *shelve* напоминает работу со словарями, которые также используют ключи для сохранения и извлечения объектов.

Для открытия файла модуль *shelve* использует функцию **open()**:

```
open(путь_к_файлу[, flag="c"[, protocol=None[, writeback=False]]])
```

где параметр *flag* может принимать значения:

- ✓ **c**: файл открывается для чтения и записи (значение по умолчанию). Если файл не существует, то он создается.
- ✓ **r**: файл открывается только для чтения.
- ✓ **w**: файл открывается для записи.
- ✓ **n**: файл открывается для записи. Если файл не существует, то он создается. Если он существует, то он перезаписывается.

Для закрытия подключения к файлу вызывается метод **close()**:

```
import shelve
d = shelve.open(filename)
d.close()
```

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11     print(states["London"])
12     print(states["Madrid"])]
```

Great Britain
Spain

Запись данных предполагает установку значения для определенного ключа:

```
states["London"] = "Great Britain"
```


А чтение из файла эквивалентно получению значения по ключу:

```
print(states["London"])
```

В качестве ключей используются строковые значения.

При чтении данных, если запрашиваемый ключ отсутствует, то генерируется исключение. В этом случае перед получением мы можем проверять на наличие ключа с помощью оператора **in**:

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11     key = "Madrid"
12     if key in states:
13         print(states[key])
14
15 Spain
```

Также мы можем использовать метод **get()**. Первый параметр метода – ключ, по которому следует получить значение, а второй – значение по умолчанию, которое возвращается, если ключ не найден

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11     state = states.get("Brussels", "Undefined")
12     print(state)
13
14 Undefined
```

Используя цикл **for**, можно перебрать все значения из файла:

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11     for key in states:
12         print(key, " - ", states[key])
```

London - Great Britain
Paris - France
Berlin - Germany
Madrid - Spain

Метод **keys()** возвращает все ключи из файла, а метод **values()** – все значения:

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11
12     for city in states.keys():
13         print(city, end=" ")
14     print()
15     for country in states.values():
16         print(country, end=" ")
17
18 London Paris Berlin Madrid
19 Great Britain France Germany Spain
```

Еще один метод **items()** возвращает набор кортежей. Каждый кортеж содержит ключ и значение.

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11
12     for state in states.items():
13         print(state)
14
15 ('London', 'Great Britain')
16 ('Paris', 'France')
17 ('Berlin', 'Germany')
18 ('Madrid', 'Spain')
```

Обновление данных

Для изменения данных достаточно присвоить по ключу новое значение, а для добавления данных – определить новый ключ:

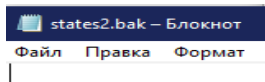
```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11
12     state = states.pop("London", "NotFound")
13     print(state)
```

Также для удаления может применяться оператор **del**:

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11
12     del states["Madrid"] # удаляем объект с ключом Madrid
```

Для удаления всех элементов можно использовать метод **clear()**:

```
1 import shelve
2
3 FILENAME = "states2"
4 with shelve.open(FILENAME) as states:
5     states["London"] = "Great Britain"
6     states["Paris"] = "France"
7     states["Berlin"] = "Germany"
8     states["Madrid"] = "Spain"
9
10 with shelve.open(FILENAME) as states:
11
12     states.clear()
```



Модуль OS и работа с файловой системой

Ряд возможностей по работе с каталогами и файлами предоставляет встроенный модуль **os**. Хотя он содержит много функций, рассмотрим только основные из них:

- ♦ **makedirs()**: создает новую папку
- ♦ **rmdir()**: удаляет папку
- ♦ **rename()**: переименовывает файл
- ♦ **remove()**: удаляет файл

Создание и удаление папки

Для создания папки применяется функция **makedirs()**, в которую передается путь к создаваемой папке:

```
1 import os
2
3 # путь относительно текущего скрипта
4 os.makedirs("Папка")
5 # абсолютный путь
6 os.makedirs("c://Папка1")
7 os.makedirs("c://Папка1/Папка")
```

Имя	Дата изменения	Тип	Размер
▼ Сегодня (9)			
Папка	15.11.2020 2:15	Папка с файлами	
Этот компьютер > Локальный диск (C:) > Папка1			
Имя	Дата изменения	Тип	
Папка	15.11.2020 2:15	Папка с файлами	

Для удаления папки используется функция **rmdir()**, в которую передается путь к удаляемой папке:

```
1 import os
2
3 # путь относительно текущего скрипта
4 os.rmdir("Папка")
5
```

Переименование файла

Для переименования вызывается функция **rename (source, target)**, первый параметр которой – путь к исходному файлу, а второй – новое имя файла. В качестве путей могут использоваться как абсолютные, так и относительные. Например, пусть в папке *C://SomeDir/* располагается файл *Text.txt*. Переименуем его в файл *"Text1.txt"*:

```
import os

os.rename("C://SomeDir/Text.txt", "C://SomeDir/Text1.txt")
```

Удаление файла

Для удаления вызывается функция **remove()**, в которую передается путь к файлу:

```
import os

os.remove("C://SomeDir/Text1.txt")
```

Пример. Посчитать количество строк в файле и количество слов и символов в каждой строке.

Метод решения.

Подсчет строк: ввести счетчик, присвоить ему 0. Пока не будет достигнут конец файла, считывать очередную строку файла и увеличивать счетчик на 1.

Подсчет символов в строке: измерять длину очередной строки с помощью встроенный в язык программирования функции.

Подсчет слов в строке:

1. Ввести счетчик слов и присвоить ему 0.
2. Ввести флаговую переменную и присвоить ей 0 (сигнал нахождения вне слова).
3. Пока не будет достигнут конец строки:
 - 1) если очередной символ не пробел и флаг указывает на нахождение вне слова, то увеличить счетчик слов и присвоить флаговой переменной 1 (сигнал нахождения внутри слова);
 - 2) если же очередной символ пробел, то присвоить флагу 0.

```
1 f = open('text.txt')
2 line = 0
3 for i in f:
4     line += 1
5
6     flag = 0
7     word = 0
8     for j in i:
9         if j != ' ' and flag == 0:
10             word += 1
11             flag = 1
12         elif j == ' ':
13             flag = 0
14
15     print(i,len(i),'симв.','word','сл.')
16
17 print(line,'стр.')
18 f.close()
```

Язык программирования Python

29 симв. 3 сл.

Объектно ориентированный язык программирования

47 симв. 4 сл.

Дистанционное обучение

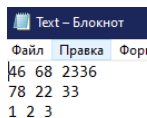
24 симв. 2 сл.

3 стр.

Пример. Сумма чисел в файле (несколько чисел в строке)

```
1 f = open('text.txt','r')
2 a=f.read().split()
3 f.close()
4 s=0
5 for el in a:
6     s+=int(el)
7 b=open('text1.txt','w')
8 b.write(str(s)+'\n')
9 print ('сумма чисел =',s)
10 b.close()
```

сумма чисел = 2589



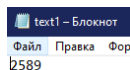
Text - Блокнот

Файл Правка Формат

46 68 2336

78 22 33

1 2 3



text1 - Блокнот

Файл Правка Формат

2589

Задачи для самостоятельной работы

1. Дан файл f , компоненты которого являются действительными числами. Найти: Сумму компонент файла f . Записать эту сумму после компоненты с номером N .
2. Дан файл f , компоненты которого являются действительными числами. Найти: Произведение компонент файла f . Записать это произведение после компоненты с номером N .
3. Дан файл f , компоненты которого являются действительными числами. Найти: Сумму квадратов компонент файла f . Записать эту сумму в конец файла N раз.
4. Дан файл f , компоненты которого являются действительными числами (положительные и отрицательные). Найти: Модуль суммы и квадрат произведения компонент файла f . Записать эти значения в конец файла.
5. Дан файл f , компоненты которого являются целыми числами. Удалить из файла f все повторные вхождения первого элемента.
6. Дан файл f , компоненты которого являются действительными числами. Найти: Наибольшее из значений компонент файла. Записать это значение после наименьшего значения компоненты файла.
7. Дан файл f , компоненты которого являются действительными числами. Найти: Наименьшее из значений компонент файла с четными номерами. Записать это значение N раз в конец файла.
8. Дан файл f , компоненты которого являются действительными числами (положительные и отрицательные). Найти: Наибольшее из значений модулей компонент файла с нечетными номерами. Записать это значение после компоненты файла с номером N .
9. Дан файл f , компоненты которого являются целыми числами. Найти: количество четных и нечетных чисел. Добавить эти значения в конец файла.
10. Дан файл f , компоненты которого являются действительными числами. Найти: Количество положительных чисел. Записать это значение в начало файла. А найденное количество отрицательных чисел записать в конец файла.
11. Дан файл f , компоненты которого являются целыми числами. Найти: количество четных чисел среди компонент с нечетными номерами. Добавить это значение в конец файла.
12. Дан файл f , компоненты которого являются целыми числами. Количество удвоенных нечетных чисел среди компонент файла. Записать это значение в начало файла.
13. Дан файл f , компоненты которого являются целыми числами. Найти количество квадратов нечетных чисел среди компонент файла. Записать это значение в начало файла.

14. Дан файл f , компоненты которого являются действительными числами. Найти: Наибольшее значение в файле. Записать это значение в конец файла столько раз, сколько положительных чисел было в исходном файле.

15. Дан файл f , компоненты которого являются целыми числами. Найти наибольшее значение в каждой десятке чисел и вставить его после этой десятки.

16. Дан файл f , компоненты которого являются действительными числами. Поменять местами наибольшее и наименьшее из этих чисел, а их сумму дописать в конец файла

17. Дан файл f , компоненты которого являются целыми числами. Найти наибольшее значение среди отрицательных (если оно есть) и вставить после последнего отрицательного числа.

18. Дан файл f , компоненты которого являются целыми числами. Найти наименьшее значение среди положительных (если оно есть) и вставить после первого положительного числа.

19. Дан файл f , компоненты которого являются целыми числами. Найти: количество четных и нечетных чисел. Добавить эти значения в конец файла.

20. Дан файл f , компоненты которого являются действительными числами. Найти: Количество положительных чисел. Записать это значение в начало файла. А найденное количество отрицательных чисел записать в конец файла

21. Дан файл f , компоненты которого являются целыми числами. Найти: количество четных чисел среди компонент с нечетными номерами. Добавить это значение в конец файла.

22. Дан файл f , компоненты которого являются целыми числами. Количество удвоенных нечетных чисел среди компонент файла. Записать это значение в начало файла.

23. Дан файл f , компоненты которого являются целыми числами. Найти количество квадратов нечетных чисел среди компонент файла. Записать это значение в начало файла.

24. Дан файл f , компоненты которого являются действительными числами. Найти: Наибольшее значение в файле. Записать это значение в конец файла столько раз, сколько положительных чисел было в исходном файле.

25. Дан файл f , компоненты которого являются целыми числами. Найти наибольшее значение в каждой десятке чисел и вставить его после этой десятки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Прохоренок Н.А. Python 3. Самое необходимое / Н.А. Прохоренок, В.А. Дронов. –2-е изд., перераб. и доп. – Санкт-Петербург: БХВ-Петербург, 2019. – 608 с.
2. Федоров Д.Ю. Программирование на языке высокого уровня Python : учеб. пособие для прикладного бакалавриата / Д.Ю. Федоров. – Москва: Издательство Юрайт, 2017. – 126 с.

ОГЛАВЛЕНИЕ

Введение	3
Лабораторная работа № 7. Преобразование символьных величин	4
Краткие теоретические сведения	4
Задачи для самостоятельной работы	17
Лабораторная работа № 8. Структуры данных. Списки. Кортежи.....	22
Краткие теоретические сведения	22
Задачи для самостоятельного решения	32
Лабораторная работа № 9. Множества	36
Краткие теоретические сведения	36
Задачи для самостоятельной работы	46
Лабораторная работа № 10. Использование функций	49
Краткие теоретические сведения	49
Задачи для самостоятельной работы	56
Лабораторная работа № 11. Регулярные выражения	60
Краткие теоретические сведения	60
Практика	69
Задачи для самостоятельной работы	70
Лабораторная работа № 12. Словари	73
Краткие теоретические сведения	73
Примеры решения задач	80
Задачи для самостоятельной работы	84
Лабораторная работа № 13. Файлы.....	87
Краткие теоретические сведения	87
Задачи для самостоятельной работы	102
Список использованных источников	104