Лабораторная работа 2.4 Работа со списками в языке Python

Цель работы: *приобретение навыков по работе со списками при написании программ с помощью языка программирования Python версии 3.x.*

Ход работы

Что такое список (*list*) в Python?

Список (*list*) – это структура данных для хранения объектов различных типов. Если вы использовали другие языки программирования, то вам должно быть знакомо понятие массива. Так вот, список очень похож на массив, только, как было уже сказано выше, в нем можно хранить объекты различных типов. Размер списка не статичен, его можно изменять. Список по своей природе является изменяемым типом данных. Переменная, определяемая как список, содержит ссылку на структуру в памяти, которая в свою очередь хранит ссылки на какие-либо другие объекты или структуры.

Создание списка в Python

Для создания списка нужно заключить элементы в квадратные скобки:

```
my_list = [1, 2, 3, 4, 5]
```

Список может выглядеть так:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
```

Можно смешивать типы содержимого:

```
my_list = ['один', 10, 2.25, [5, 15], 'пять']
```

Поддерживаются вложенные списки как в примере выше.

Получать доступ к любому элементу списка можно через его индекс. В Python используется система индексации, начиная с нуля.

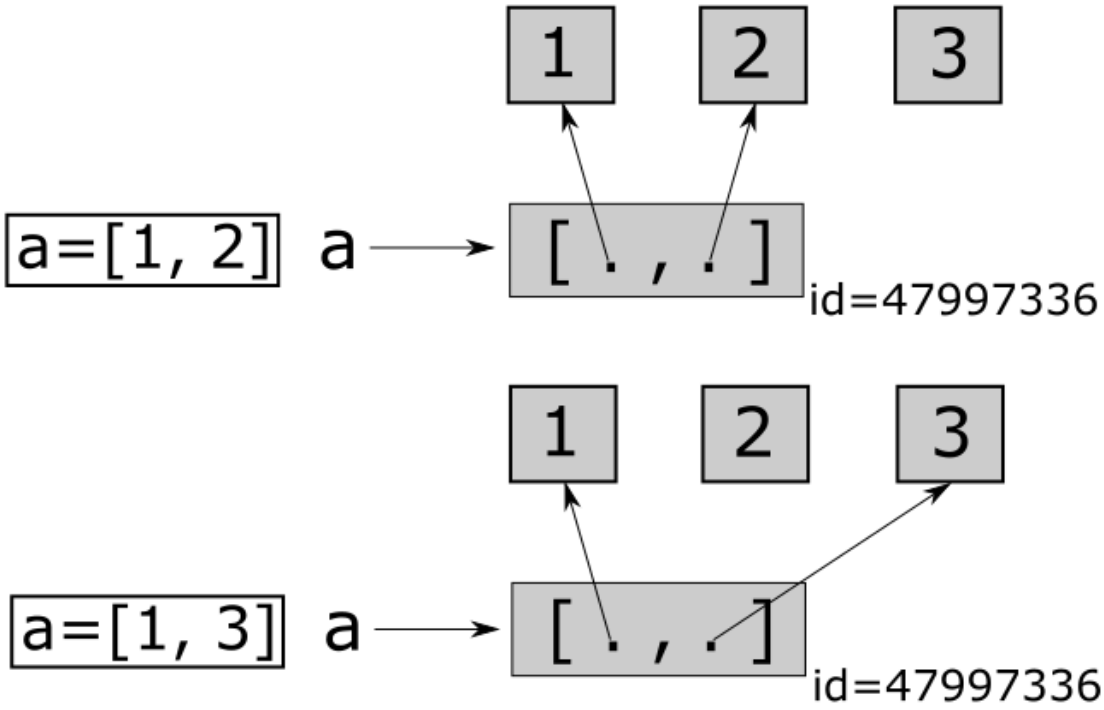
```
third_elem  =  my_list[2]
```

Принцип похож на строки.

Как списки хранятся в памяти?

Как уже было сказано выше, список является изменяемым типом данных. При его создании в памяти резервируется область, которую можно условно назвать некоторым “контейнером”, в котором хранятся ссылки на другие элементы данных в памяти. В отличии от таких типов данных как число или строка, содержимое “контейнера” списка можно менять. Для того, чтобы лучше визуально представлять себе этот процесс взгляните на картинку ниже. Изначально был создан

список содержащий ссылки на объекты 1 и 2, после операции *a[1] = 3*, вторая ссылка в списке стала указывать на объект 3.



Проход (итерация) по списку

Читать элементы списка можно с помощью следующего цикла:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
for elem in my_list:
    print(elem)
```

Таким образом можно читать элементы списка. А вот что касается их обновления:

```
my_list = [1, 2, 3, 4, 5]
for i in range(len(my_list)):
    my_list[i]+=5
    print(my_list)
```

Результат будет следующим:

```
[6, 7, 8, 9, 10]
```

Функция `len()` используется для возврата количества элементов.

Стоит запомнить, что вложенный список — это всегда один элемент вне зависимости от количества его элементов.

```
my_list = ['один', 10, 2.25, [5, 15], 'пять']
print(len(my_list))
```

Результат кода выше — `5`.

Для получения в цикле одновременно элемента списка и его индекса необходимо использовать функцию `enumerate`. Встроенная в Python функция enumerate() применяется для итерируемых коллекций (строки, списки, словари и т. д.) и создает объект, который генерирует кортежи, состоящие из двух элементов - индекса элемента и самого элемента.

```
>>> a = [10, 20, 30, 40]
>>> for i, item in enumerate(a):
...     print(f"({i}, {item})")
...
(0, 10)
(1, 20)
(2, 30)
(3, 40)
```

Арифметические операции со списками

Для объединения списков можно использовать оператор сложения (`+`):

```
list_1 = [1, 2, 3]
list_2 = [4, 5, 6]
print(list_1 + list_2)
```

Результат:

```
[1, 2, 3, 4, 5, 6]
```

Список можно повторить с помощью оператора умножения (`*`):

```
list_1 = [1, 2, 3]
print(list_1  *  2)
```

Результат:

```
[1, 2, 3, 1, 2, 3]
```

Поиск элемента в списке Python

Для того, чтобы проверить, есть ли заданный элемент в списке Python необходимо использовать оператор `in`:

```
lst = [3, 5, 2, 4, 1]
if 3 in lst:
    print("Список содержит число 3")
else:
    print("Список не содержит число 3")
```

Результат:

```
Список содержит число 3
```

Если требуется, чтобы элемент отсутствовал в списке, необходимо использовать оператор `not in`:

```
lst = [3, 5, 2, 4, 1]
if 0 not in lst:
    print("Список не содержит нулей")
```

Результат:

```
Список не содержит нулей
```

Индекс элемента в списке

Метод *index* можно использовать для получения индекса элемента:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
print(my_list.index('два'))
```

Результат `1`.

Если в списке больше одного такого же элемента, функция вернет индекс первого.

Число вхождений элемента в список

Метод *count* можно использовать для определения числа сколько раз данный элемент встречается в списке:

```
lst = [1, 2, 2, 3, 3]
print(lst.count(2))
```

Результат `2`.

Изменение списка Python

Списки — это изменяемые объекты, поэтому их элементы могут изменяться, или же может меняться их порядок. Если есть такой список:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
```

То его третий элемент можно изменить следующим образом:

```
my_list[2]  =  'ноль'
```

Если сейчас вывести его на экран, то он будет выглядеть вот так:

```
['один', 'два', 'ноль', 'четыре', 'пять']
```

Если индекс — отрицательное число, то он будет считаться с последнего элемента.

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
elem  =  my_list[-1]
print(elem)
```

Вывод этого кода — ‘пять’.

Вставить элемент в список

Метод *insert* можно использовать, чтобы вставить элемент в список:

```
my_list = [1, 2, 3, 4, 5]
my_list.insert(1,'Привет')
print(my_list)
```

Результат:

```
[1, 'Привет', 2, 3, 4, 5]
```

Индексы для вставляемых элементов также начинаются с нуля.

Добавить элемент в список

Метод *append* можно использовать для добавления элемента в список:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
my_list.append('ещё один')
print(my_list)
```

Результат:

```
['один', 'два', 'три', 'четыре', 'пять', 'ещё один']
```

Можно добавить и больше одного элемента таким способом:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
list_2  =  ['шесть',  'семь']
my_list.extend(list_2)
print(my_list)
```

Результат:

```
['один', 'два', 'три', 'четыре', 'пять', 'шесть',  'семь']
```

При этом `list_2` не поменяется.

Отсортировать список

Для сортировки списка нужно использовать метод *sort*.

```
my_list = ['cde', 'fgh', 'abc', 'klm', 'opq']
list_2 = [3, 5, 2, 4, 1]
my_list.sort()
list_2.sort()
print(my_list)
print(list_2)
```

Вывод:

```
['abc', 'cde', 'fgh', 'klm', 'opq']
[1, 2, 3, 4, 5]
```

Для сортировки списка в порядке убывания необходимо вызвать метод *sort* с аргументом *reverse=True*.

```
list_2 = [3, 5, 2, 4, 1]
list_2.sort(reverse=True)
```

Вывод:

```
[5, 4, 3, 2, 1]
```

Перевернуть список

Можно развернуть порядок элементов в списке с помощью метода *reverse*:

```
my_list = [1, 2, 3, 4, 5]
my_list.reverse()
print(my_list)
```

Результат:

```
[5, 4, 3, 2, 1]
```

Удалить элемент из списка

Удалить элемент можно, написав его индекс в методе *pop*:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
removed = my_list.pop(2)
print(my_list)
print(removed)
```

Результат:

```
['один', 'два', 'четыре', 'пять']
три
```

Если не указывать индекс, то функция удалит последний элемент.

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
removed = my_list.pop()
print(my_list)
print(removed)
```

Результат:

```
['один', 'два', 'три', 'четыре']
пять
```

Элемент можно удалить с помощью метода *remove*.

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
my_list.remove('два')
print(my_list)
```

Результат:

```
['один', 'три', 'четыре', 'пять']
```

Оператор `del` можно использовать для тех же целей:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
del my_list[2]
print(my_list)
```

Результат:

```
['один', 'два', 'четыре', 'пять']
```

Можно удалить несколько элементов с помощью оператора среза:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
del my_list[1:3]
print(my_list)
```

Результат:

```
['один', 'четыре', 'пять']
```

Удалить все элементы из списка

Можно удалить все элементы из списка с помощью метода *clear*:

```
a = [1, 2, 3, 4, 5]
print(a)
a.clear()
print(a)
```

Результат

```
[1, 2, 3, 4, 5]
[]
```

List Comprehensions

List Comprehensions чаще всего на русский язык переводят как абстракция списков или списковое включение, является частью синтаксиса языка, которая предоставляет простой способ построения списков. Проще всего работу *list comprehensions* показать на примере. Допустим вам необходимо создать список целых чисел от *0* до *n*, где *n* предварительно задается. Классический способ решения данной задачи выглядел бы так:

```
>>> n = int(input())
7
>>> a=[]
>>> for i in range(n):
        a.append(i)

>>> print(a)
[0, 1, 2, 3, 4, 5, 6]
```

Использование *list comprehensions* позволяет сделать это значительно проще:

```
>>> n = int(input())
7
>>> a = [i for i in range(n)]
>>> print(a)
[0, 1, 2, 3, 4, 5, 6]
```

или вообще вот так, в случае если вам не нужно больше использовать *n:*

```
>>> a = [i for i in range(int(input()))]
7
>>> print(a)
[0, 1, 2, 3, 4, 5, 6]
```

List Comprehensions как обработчик списков

В языке *Python* есть две очень мощные функции для работы с коллекциями: *map* и *filter*. Они позволяют использовать функциональный стиль программирования, не прибегая к помощи циклов, для работы с такими типами как *list*, *tuple*, *set*, *dict* и т.п. Списковое включение позволяет обойтись без этих функций. Приведем несколько примеров для того, чтобы понять о чем идет речь.

Пример с заменой функции *map*.

Пусть у нас есть список и нужно получить на базе него новый, который содержит элементы первого, возведенные в квадрат. Решим эту задачу с использованием циклов:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = []
>>> for i in a:
        b.append(i**2)
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

Та же задача, решенная с использованием *map*, будет выглядеть так:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = list(map(lambda x: x**2, a))
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

В данном случае применена *lambda*-функция, о том, что это такое и как ее использовать можно узнать из лабораторной работы, посвященной функциям.

Через списковое включение эта задача будет решена так:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = [i**2 for i in a]
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

Пример с заменой функции *filter*.

Построим на базе существующего списка новый, состоящий только из четных чисел:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = []
>>> for i in a:
        if i%2 == 0:
            b.append(i)
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Решим эту задачу с использованием *filter*:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = list(filter(lambda x: x % 2 == 0, a))
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Решение через списковое включение:

```
>>> a = [1, 2, 3, 4, 5, 6, 7]
>>> b = [i for i in a if i % 2 == 0]
>>> print('a = {}\nb = {}'.format(a, b))
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Слайсы / Срезы

Слайсы (срезы) являются очень мощной составляющей *Python*, которая позволяет быстро и лаконично решать задачи выборки элементов из списка. Выше уже был пример использования слайсов, здесь разберем более подробно работу с ними. Создадим список для экспериментов:

```
>>> a = [i for i in range(10)]
```

Слайс задается тройкой чисел, разделенных запятой: *start:stop:step*. *Start* – позиция с которой нужно начать выборку, *stop* – конечная позиция, *step* – шаг. При этом необходимо помнить, что выборка не включает элемент определяемый *stop*.

Рассмотрим примеры:

```
>>> # Получить копию списка
>>> a[:]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> # Получить первые пять элементов списка
>>> a[0:5]
[0, 1, 2, 3, 4]

>>> # Получить элементы с 3-го по 7-ой
>>> a[2:7]
[2, 3, 4, 5, 6]

>>> # Взять из списка элементы с шагом 2
>>> a[::2]
[0, 2, 4, 6, 8]

>>> # Взять из списка элементы со 2-го по 8-ой с шагом 2
>>> a[1:8:2]
[1, 3, 5, 7]
```

Слайсы можно сконструировать заранее, а потом уже использовать по мере необходимости. Это возможно сделать, в виду того, что слайс – это объект класса *slice*. Ниже приведен пример, демонстрирующий эту функциональность:

```
>>> s = slice(0, 5, 1)
>>> a[s]
[0, 1, 2, 3, 4]

>>> s = slice(1, 8, 2)
>>> a[s]
[1, 3, 5, 7]
```

Функции агрегации

Для работы со списками Python предоставляет следующие функции:

- `len(L)` - получить число элементов в списке `L`.
- `min(L)` - получить минимальный элемент списка `L`.
- `max(L)` - получить максимальный элемент списка `L`.
- `sum(L)` - получить сумму элементов списка `L`, если список `L` содержит только числовые значения.

Для функций `min` и `max` элементы списка должны быть сравнимы между собой.

Например:

```
>>> my_list = [5, 3, 2, 4, 1]
>>> print(len(my_list))
5
>>> print(min(my_list))
1
>>> print(max(my_list))
5
>>> print(sum(my_list))
15
```

Сравнение списков

В Python 2 сравнить элементы двух списком можно с помощью функции `cmp`:

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
list_2 = ['три', 'один', 'пять', 'два', 'четыре']
print(cmp(my_list,list_2))
```

Она вернет `-1`, если списки не совпадают, и `1` в противном случае.

В Python 3 для этого используется оператор (`==`):

```
my_list = ['один', 'два', 'три', 'четыре', 'пять']
list_2 = ['три', 'один', 'пять', 'два', 'четыре']
if (my_list == list_2):
    print('совпадают')
else:
    print('не совпадают')
```

Результат `не совпадают`.

Списки и строки

Для конвертации строки в набор символов, можно использовать функцию `list`:

```
my_str = 'Monty Python'
my_list = list(my_str)
print(my_list)
```

Результат:

```
['M', 'o', 'n', 't', 'y', ' ', 'P', 'y', 't', 'h', 'o', 'n']
```

Функция `list` используется для того, чтобы разбивать строку на отдельные символы.

Можно использовать метод *split* для разбиения строки на слова:

```
my_str = 'Monty Python'
my_list = my_str.split()
print(my_list)
```

Результат:

```
['Monty', 'Python']
```

Она возвращает обычный список, где с каждым словом можно взаимодействовать через индекс.

Символом разбиения может служить любой знак, а не только пробел.

```
my_str = 'Monty-Python'
my_list = my_str.split('-')
print(my_list)
```

Результат будет аналогичен:

```
['Monty', 'Python']
```

Обратный процесс — объединение элементов списка в строку. Это делается с помощью метода `join`:

```
my_list = ['Monty', 'Python']
delimiter = ' '
output = delimiter.join(my_list)
print(output)
```

Результат `Monty Python`.

Алиасинг (псевдонимы)

Когда две переменные ссылаются на один и тот же объект:

```
my_list = ['Monty', 'Python']
list_2 = my_list
```

Алиасинг значит, что на объект ссылается больше одного имени.

Следующий пример показывает, как меняются изменяемые списки:

```
my_list = ['Monty', 'Python']
list_2 = my_list
list_2[1] = 'Java:)'
print(my_list)
```

Результат:

```
['Monty', 'Java:)']
```

Изменился список `list_2`, но поскольку он ссылается на один и тот же объект, то оригинальный список тоже поменялся. Использовать “псевдонимы” при работе со списками не рекомендуется. Поэтому для создания копии списка необходимо использовать либо метод *copy*, либо использовать оператор среза.

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a.copy()
>>> b
[1, 2, 3, 4, 5]
>>> a == b
True
>>> a is b
False
>>> a is not b
True
```

Пример 1. *Ввести список A из 10 элементов, найти сумму элементов, меньших по модулю 5, и вывести ее на экран.*

Решение: Напишем программу для решения поставленной задачи.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys


if __name__ == '__main__':
    # Ввести список одной строкой.
    A = list(map(int, input().split()))
    # Проверить количество элементов списка.
    if len(A) != 10:
        print("Неверный размер списка", file=sys.stderr)
        exit(1)

    # Найти искомую сумму.
    s = 0
    for item in A:
        if abs(item) < 5:
            s += item

    print(s)
```

Выражение `list(map(int, input().split()))` позволяет ввести целочисленный массив одной строкой. Это работает, поскольку изначально строка, полученная с помощью функции `input()` разбивается на список подстрок с помощью метода `split()`. Затем к каждой из подстрок списка функция `map` применяет функцию `int` (точнее вызов `int` создает целое число на основании значения своего аргумента). Поскольку функция `map` возвращает генератор, то с помощью вызова `list` генератор преобразуется к списку.

Данная задача может быть также решена с помощью списковых включений следующим образом:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys


if __name__ == '__main__':
    # Ввести список одной строкой.
    A = list(map(int, input().split()))
    # Проверить количество элементов списка.
    if len(A) != 10:
        print("Неверный размер списка", file=sys.stderr)
        exit(1)

    # Найти искомую сумму.
    s = sum([a for a in A if abs(a) < 5])
    print(s)
```

Функция `abs` в этом примере позволяет получить абсолютное значение целого числа. Для вещественных чисел лучше использовать функцию `fabs` пакета `math`.

Пример 2. *Написать программу, которая для целочисленного списка определяет, сколько положительных элементов располагается между его максимальным и минимальным элементами.*

Решение: Запишем алгоритм в самом общем виде.

1. Определить, где в массиве расположены его максимальный и минимальный элементы, то есть найти их индексы.
2. Просмотреть все элементы, расположенные между ними. Если элемент массива больше нуля, увеличить счетчик элементов на единицу.

Ясно, что порядок расположения элементов в массиве заранее не известен, и сначала может следовать как максимальный, так и минимальный элемент, кроме того, они могут и совпадать. Поэтому прежде чем просматривать массив в поисках количества положительных элементов, требуется определить, какой из этих индексов больше. Запишем уточненный алгоритм:

1. Определить, где в массиве расположены его максимальный и минимальный элементы:

- задать начальные значения для индексов максимального и минимального элементов (например, равные нулю, но можно использовать любые другие значения индекса, не выходящие за границу массива).
- просмотреть массив, поочередно сравнивая каждый его элемент с ранее найденными максимумом и минимумом. Если очередной элемент больше ранее найденного максимума, принять этот элемент за новый максимум (т. е. запомнить его индекс). Если очередной элемент меньше ранее найденного минимума, принять этот элемент за новый минимум.

2. Определить границы просмотра массива для поиска положительных элементов, находящихся между его максимальным и минимальным элементами:

- если максимум расположен в массиве раньше, чем минимум, принять левую границу просмотра равной индексу максимума, иначе - индексу минимума.
- если максимум расположен в массиве раньше, чем минимум, принять правую границу просмотра равной индексу минимума, иначе - индексу максимума.

3. Обнулить счетчик положительных элементов. Просмотреть массив в указанном диапазоне. Если очередной элемент больше нуля, увеличить счетчик на единицу.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys


if __name__ == '__main__':
    # Ввести список одной строкой.
    a = list(map(int, input().split()))
    # Если список пуст, завершить программу.
    if not a:
        print("Заданный список пуст", file=sys.stderr)
        exit(1)

    # Определить индексы минимального и максимального элементов.
    a_min = a_max = a[0]
    i_min = i_max = 0
    for i, item in enumerate(a):
        if item < a_min:
            i_min, a_min = i, item
```

```
        if item >= a_max:
            i_max, a_max = i, item

    # Проверить индексы и обменять их местами.
    if i_min > i_max:
        i_min, i_max = i_max, i_min

    # Посчитать количество положительных элементов.
    count = 0
    for item in a[i_min+1:i_max]:
        if item > 0:
            count += 1

    print(count)
```

Следует обратить внимание как осуществляется поиск индексов минимального и максимального элементов. В простейшем случае этот поиск можно было бы осуществить следующим образом:

```
i_min = a.index(min(a))
i_max = a.index(max(a))
```

Однако, в этом случае список будет просмотре четыре раза, тогда как в данном коде список будет просмотрен только один раз, что увеличивает скорость выполнения программы, особенно при работе с большими списками.

Аппаратура и материалы

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub, рекомендован к использованию Google Chrome.
5. Дистрибутив языка программирования Python, включающий набор популярных библиотек Anaconda.
6. Интегрированная среда разработки PyCharm Community Edition.

Указания по технике безопасности

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;
- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.

4. Дополните файл `.gitignore` необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы. Создайте для каждого примера отдельный модуль языка Python. Зафиксируйте изменения в репозитории.
8. Приведите в отчете скриншоты результатов выполнения каждой из программ примеров при различных исходных данных вводимых с клавиатуры.
9. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.
10. Зафиксируйте сделанные изменения в репозитории.
11. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.
12. Выполните слияние ветки для разработки с веткой *main* / *master*.
13. Отправьте сделанные изменения на сервер GitHub.
14. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Индивидуальные задания

Задание 1

Составить программу с использованием одномерных массивов для решения задачи. Номер варианта необходимо получить у преподавателя. Решить индивидуальное задание как с использованием циклов, так и с использованием List Comprehensions.

1. Ввести список *A* из 10 элементов, найти наибольший элемент и переставить его с первым элементом. Преобразованный массив вывести.
2. Ввести список *A* из 10 элементов, найти произведение положительных элементов и вывести его на экран.
3. Ввести список *A* из 10 элементов, найти наименьший элемент и переставить его с последним элементом. Преобразованный массив вывести.
4. Ввести список *A* из 10 элементов, найти сумму отрицательных элементов и вывести ее на экран.
5. Ввести список *A* из 10 элементов, найти сумму элементов, больших 3 и меньших 8 и вывести ее на экран.
6. Ввести список *A* из 10 элементов, найти разность положительных элементов, и вывести ее на экран.
7. Ввести список *A* из 10 элементов, найти произведение отрицательных элементов и вывести его на экран.
8. В заданном список подсчитать число нулевых элементов и вывести на экран их индексы.
9. Составить программу, выдающую индексы заданного элемента или сообщающую, что такого элемента в списке нет.
10. Ввести список *A* из 10 элементов, найти произведение положительных элементов кратных 3, их количество и вывести результаты на экран.
11. Ввести список *A* из 10 элементов, найти сумму отрицательных элементов кратных 7, их количество и вывести результаты на экран.
12. Ввести список *A* из 10 элементов, найти сумму элементов, больших 2 и меньших 20 и кратных 8, их количество и вывести результаты на экран.
13. Ввести список *A* из 10 элементов, найти сумму элементов, меньших по модулю 3 и кратных 9, их количество и вывести результаты на экран.
14. Ввести список *A* из 10 элементов, найти разность положительных элементов кратных 11, их количество и вывести результаты на экран.
15. Ввести список *A* из 10 элементов, найти произведение элементов, больших 8 и меньших 18 и кратных 10, их количество и вывести результаты на экран.

16. Ввести список *A* из 10 элементов, найти сумму элементов кратных 2, их количество и вывести результаты на экран.
17. Ввести список *A* из 10 элементов, найти квадраты элементов кратных 4 и их количество. Преобразованный массив вывести.
18. Ввести список *A* из 10 элементов, найти сумму положительных элементов кратных 5, их количество и вывести результаты на экран.
19. Ввести список *A* из 10 элементов. Определить количество элементов, кратных 3 и индексы последнего такого элемента.
20. В списках *U*, *D*, *V* содержатся значения утренней, дневной и вечерней температуры соответственно за каждый день недели. Подсчитать среднее значение дневной температуры за каждый день.
21. В списках *A*, *G*, *F* содержатся оценки учащихся по алгебре, геометрии и физике соответственно. Определить, по какому предмету лучше успеваемость.
22. В списках *U*, *D*, *V* содержатся значения утренней, дневной и вечерней температуры соответственно за каждый день недели. Сформировать список S, в котором будут содержаться значения среднедневной температуры. Определить среднее значение температуры а неделю.
23. В списках *A*, *G*, *F* содержатся оценки учащихся по алгебре, геометрии и физике соответственно. Определить среднюю оценку по алгебре и количество учащихся, не имеющих ни одной «двойки».
24. Определить средний рост девочек и мальчиков одного класса. В классе учится n учеников. ($n > 15$).
25. В ЭВМ по очереди поступают результаты соревнований по плаванию на дистанции 200 м, в которых участвует n спортсменов ($n > 10$). Выдать на экран дисплея лучший результат.
26. Из списка целых чисел составить три других, в первый из которых записать числа, кратные 5, во второй - числа, кратные 7, а в третий - остальные числа.
27. Для заданного списка определить, каких элементов больше: положительных или отрицательных. Вывести на экран их количество.
28. В заданном списке все отрицательные элементы заменить нулями и подсчитать их количество.

Задание 2

Составить программу с использованием одномерных массивов для решения задачи на переупорядочивание элементов массива. Для сортировки допускается использовать метод *sort* с заданным параметром *key* (https://docs.python.org/3/howto/sorting.html) и объединение нескольких списков. Номер варианта необходимо получить у преподавателя.

1. В списке, состоящем из вещественных элементов, вычислить:

1. сумму отрицательных элементов списка;
2. произведение элементов списка, расположенных между максимальным и минимальным элементами.

Упорядочить элементы списка по возрастанию.

2. В списке, состоящем из вещественных элементов, вычислить:

1. сумму положительных элементов списка;
2. произведение элементов списка, расположенных между максимальным по модулю и минимальным по модулю элементами.

Упорядочить элементы списка по убыванию.

3. В списке, состоящем из целых элементов, вычислить:

1. произведение элементов списка с четными номерами;
2. сумму элементов списка, расположенных между первым и последним нулевыми элементами.

Преобразовать список таким образом, чтобы сначала располагались все положительные элементы, а потом - все отрицательные (элементы, равные 0, считать положительными).

4. В списке, состоящем из вещественных элементов, вычислить:

1. сумму элементов списка с нечетными номерами;
2. сумму элементов списка, расположенных между первым и последним отрицательными элементами.

Сжать список, удалив из него все элементы, модуль которых не превышает 1. Освободившиеся в конце списка элементы заполнить нулями.

5. В списке, состоящем из вещественных элементов, вычислить:

1. максимальный элемент списка;
2. сумму элементов списка, расположенных до последнего положительного элемента.

Сжать список, удалив из него все элементы, модуль которых находится в интервале [*a*, *b*]. Освободившиеся в конце списка элементы заполнить нулями.

6. В списке, состоящем из целых элементов, вычислить:

1. номер максимального элемента списка;
2. произведение элементов списка, расположенных между первым и вторым нулевыми элементами.

Преобразовать список таким образом, чтобы в первой его половине располагались элементы, стоявшие в нечетных позициях, а во второй половине - элементы, стоявшие в четных позициях.

7. В списке, состоящем из вещественных элементов, вычислить:

1. номер минимального элемента списка;
2. сумму элементов списка, расположенных между первым и вторым отрицательными элементами.

Преобразовать список таким образом, чтобы сначала располагались все элементы, модуль которых не превышает 1, а потом - все остальные.

8. В списке, состоящем из вещественных элементов, вычислить:

1. максимальный по модулю элемент списка;
2. сумму элементов списка, расположенных между первым и вторым положительными элементами.

Преобразовать список таким образом, чтобы элементы, равные нулю, располагались после всех остальных.

9. В списке, состоящем из целых элементов, вычислить:

1. минимальный по модулю элемент списка;
2. сумму модулей элементов списка, расположенных после первого элемента, равного нулю.

Преобразовать список таким образом, чтобы в первой его половине располагались элементы, стоявшие в четных позициях, а во второй половине - элементы, стоявшие в нечетных позициях.

10. В списке, состоящем из вещественных элементов, вычислить:

1. номер минимального по модулю элемента списка;

2. сумму модулей элементов списка, расположенных после первого отрицательного элемента.

Сжать список, удалив из него все элементы, величина которых находится в интервале [*a*, *b*]. Освободившиеся в конце списка элементы заполнить нулями.

11. В списке, состоящем из вещественных элементов, вычислить:

1. номер максимального по модулю элемента списка;
2. сумму элементов списка, расположенных после первого положительного элемента.

Преобразовать список таким образом, чтобы сначала располагались все элементы, целая часть которых лежит в интервале [*a*, *b*], а потом - все остальные.

12. В списке, состоящем из вещественных элементов, вычислить:

1. количество элементов списка, лежащих в диапазоне от *A* до *B*;
2. сумму элементов списка, расположенных после максимального элемента.

Упорядочить элементы списка по убыванию модулей элементов.

13. В списке, состоящем из вещественных элементов, вычислить:

1. количество элементов списка, равных 0;
2. сумму элементов списка, расположенных после минимального элемента.

Упорядочить элементы списка по возрастанию модулей элементов.

14. В списке, состоящем из вещественных элементов, вычислить:

1. количество элементов списка, больших *C*;
2. произведение элементов списка, расположенных после максимального по модулю элемента.

Преобразовать список таким образом, чтобы сначала располагались все отрицательные элементы, а потом - все положительные (элементы, равные 0, считать положительными).

15. В списке, состоящем из вещественных элементов, вычислить:

1. количество отрицательных элементов списка;
2. сумму модулей элементов списка, расположенных после минимального по модулю элемента.

Заменить все отрицательные элементы списка их квадратами и упорядочить элементы списка по возрастанию.

16. В списке, состоящем из вещественных элементов, вычислить:

1. количество положительных элементов списка;
2. сумму элементов списка, расположенных после последнего элемента, равного нулю.

Преобразовать список таким образом, чтобы сначала располагались все элементы, целая часть которых не превышает 1, а потом - все остальные.

17. В списке, состоящем из вещественных элементов, вычислить:

1. количество элементов списка, меньших *C*;
2. сумму целых частей элементов списка, расположенных после последнего отрицательного элемента.

Преобразовать список таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального не более чем на 20%, а потом - все остальные.

18. В списке, состоящем из вещественных элементов, вычислить:

1. произведение отрицательных элементов списка;
2. сумму положительных элементов списка, расположенных до максимального элемента.

Изменить порядок следования элементов в списке на обратный.

19. В списке, состоящем из вещественных элементов, вычислить:

1. произведение положительных элементов списка;
2. сумму элементов списка, расположенных до минимального элемента.

Упорядочить по возрастанию отдельно элементы, стоящие на четных местах, и элементы, стоящие на нечетных местах.

Содержание отчета и его форма

Отчет по лабораторной работе оформляется электронно в формате PDF, должен содержать ответы на контрольные вопросы, ссылку на репозиторий с которым выполнялась работа, скриншоты IDE PyCharm, скриншоты результатов работы программ.

Вопросы для защиты работы

1. Что такое списки в языке Python?
2. Как осуществляется создание списка в Python?
3. Как организовано хранение списков в оперативной памяти?
4. Каким образом можно перебрать все элементы списка?
5. Какие существуют арифметические операции со списками?
6. Как проверить есть ли элемент в списке?
7. Как определить число вхождений заданного элемента в списке?
8. Как осуществляется добавление (вставка) элемента в список?
9. Как выполнить сортировку списка?
10. Как удалить один или несколько элементов из списка?
11. Что такое списковое включение и как с его помощью осуществлять обработку списков?
12. Как осуществляется доступ к элементам списков с помощью срезов?
13. Какие существуют функции агрегации для работы со списками?
14. Как создать копию списка?
15. Самостоятельно изучите функцию *sorted* языка Python. В чем ее отличие от метода *sort* списков?