Списки Python.

Цель: Изучение способов добавления и удаление элементов в массив. Проверка списка на определённые элементы.

Задачи: Создание массивов. Изучить методы для изменения элементов в списках.

Теоретическая часть.

Списки в Python - упорядоченные изменяемые коллекции объектов произвольных типов (почти как массив, но типы могут отличаться). Чтобы использовать списки, их нужно создать. Создать список можно несколькими способами. Например, можно обработать любой итерируемый объект (например, строку) встроенной функцией list:

```
print(list('список'))
```

Выходные данные:

>>> ['с', 'п', 'и', 'с', 'о', 'к']

Чтобы создать список потребуются квадратные скобки и его название:

```
my_spisok = [13, 'Hello!', 0, -44, 999, '001 01 11']
print(my_spisok)
```

Выходные данные:

>>> [13, 'Hello!', 0, -44, 999, '001 01 11']

Что если в дальнейшем, при работе с данным списком нам понадобиться убрать или добавить в него элемент?

В Python есть несколько методов для удаления элементов из списка: remove(), pop() и clear(). Помимо них также существует ключевое слово del.

Методы удаления.

1) Метод remove()

Метод remove() — это встроенный метод, который удаляет первый совпадающий элемент из списка.

Синтаксис: list.remove(element).

Передается элемент, который нужно удалить из списка. Метод не возвращает значений.

Как использовать:

- Если в списке есть повторяющиеся элементы, первый совпадающий будет удален.
- Если элемента нет, будет брошена ошибка с сообщением о том, что элемент не найден.
- Метод не возвращает значений.
- В качестве аргумента нужно передать валидное значение.

Пример:

```
my_list = [3, 8, 0, 'Hi!', 10, 'Python']
print(my_list)

my_list.remove(0)
print(my_list)

my_list.remove('Hi!')
print(my_list)
```

Выходные данные:

>>> [3, 8, 0, 'Hi!', 10, 'Python']

>>> [3, 8, 'Hi!', 10, 'Python']

>>> [3, 8, 10, 'Python']

2) Метод pop()

Этот метод удаляет элемент на основе переданного индекса.

Синтаксис: list.pop(index)

Принимает лишь один аргумент — индекс.

Как использовать:

- Для удаления элемента списка нужно передать его индекс.
- Индексы в списках стартуют с 0. Для получения первого передайте 0. Для удаления последнего передайте -1.
- Этот аргумент не является обязательным. Значение по умолчанию равно -1, поэтому по умолчанию будет удален последний элемент.
- Если этот индекс не найден или он вне диапазона, то метод выбросит исключение IndexError: pop index.

Пример:

```
my_list = [3, 8, 0, 'Hi!', 10, 'Python']
print(my_list)

my_list.pop(0)
print(my_list)

my_list.pop(0)
```

```
print(my_list)

my_list.pop(2)
print(my_list)
```

Выходные данные:

>>> [3, 8, 0, 'Hi!', 10, 'Python']

>>> [8, 0, 'Hi!', 10, 'Python']

>>> [0, 'Hi!', 10, 'Python']

>>> [0, 'Hi!', 'Python']

3) Метод clear()

Метод clear() удаляет все элементы из списка.

Синтаксис: list.clear().

Нет ни параметров, ни возвращаемого значения.

Пример:

```
my_list = [3, 8, 0, 'Hi!', 10, 'Python']
print(my_list)

my_list.clear()
print(my_list)
```

Выходные данные:

>>> [3, 8, 0, 'Hi!', 10, 'Python']

>>> []

4) Метод del()

Для удаления элемента из списка можно использовать ключевое слово del с названием списка после него.

Также потребуется передать индекс того элемента, который нужно удалить.

Синтаксис: del list[index]

Также можно выбрать элементы в определенном диапазоне и удалить их с помощью del. Для этого нужно передать начальное и конечное значение диапазона.

Синтаксис: del list[start:stop]

Пример:

```
my_list = [3, 8, 0, 'Hi!', 10, 'Python']
print(my_list)

del my_list[3]
print(my_list)

del my_list[0:3]
print(my_list)
```

Выходные данные:

>>> [3, 8, 0, 'Hi!', 10, 'Python']

>>> [3, 8, 0, 10, 'Python']

>>> [10, 'Python']

Методы добавления.

1) **Метод append()**

С помощью этого метода вы можете добавить один элемент в конец списка.

Синтаксис: list.append(element)

Метод append() принимает только один аргумент - это элемент, который вы хотите добавить. Этот элемент может быть любого типа данных.

Пример:

```
numb_10 = 10
say = 'Hello!'
list1 = [10, 20, 30, 40, 50]

my_list = []
print(my_list)

my_list.append(numb_10)
print(my_list)

my_list.append(say)
print(my_list)

my_list.append(list1)
print(my_list)
```

Выходные данные:

>>> []

>>> [10]

>>> [10, 'Hello!']

>>> [10, 'Hello!', [10, 20, 30, 40, 50]]

2) **Метод insert(idx, x)**

Синтаксис: list.insert(idx, x)

idx: Позиция (индекс), на которую требуется поместить элемент. Нумерация ведётся с нуля. Поддерживается отрицательная индексация.

x: Элемент, который требуется поместить в список.

insert(idx, x) – позволяет добавить элемент в список на определённую позицию.

Пример:

```
my_list = ['начало', 'конец']
print(my list)
# вставим "середину", на свою позицию
my_list.insert(1, 'середина')
print(my_list)
```

Выходные данные:

>>> ['начало', 'конец']

>>> ['начало', 'середина', 'конец']

Добавление и удаление элементов.

Создадим 2 списка и обменяемся элементами так, чтобы в одном списке был только (строковый тип) str тип, а в другом только (числовой тип) int тип.

```
stroki = ['str1', 'str2', 1]
numbers = [2, 'str3', 3]
print('До обмена: ')
print(stroki)
print(numbers)

stroki.remove(1)
numbers.append(1)

numbers.remove('str3')
stroki.append('str3')

print('После обмена: ')
print(stroki)
print(numbers)
```

Выходные данные:

>>> До обмена:

>>> ['str1', 'str2', 1]

>>> [2, 'str3', 3]

>>> После обмена:

>>> ['str1', 'str2', 'str3']

>>> [2, 3, 1]

Далее, с помощью метода сортировки sort(), выведем элементы в порядке возрастания.

Метод sort() в Python упорядочивает элементы списка в порядке возрастания. Это естественный способ сортировки элементов.

Если вы хотите, чтобы сортировка выполнялась в обратном порядке, передайте обратный аргумент, как reverse = True. Мы можем использовать это для сортировки списка чисел в порядке убывания.

Пример с тем же списком:

```
print('После сортировки: ')
numbers.sort()
print(numbers)
print('Сортировка в обратном порядке: ')
numbers.sort(reverse=True)
print(numbers)
```

Выходные данные:

>>> После сортировки:

>>> [1, 2, 3]

>>> Сортировка в обратном порядке:

>>> [3, 2, 1]

Практическая часть.

Задание 1.

Дан массив:

m = ['круг', 0.25, 'квадрат', 'треугольник', 15, 'круг', 'овал', '10']

Оставить в нём только названия фигур. Выполнить задание, с помощью метода remove().

Задание 2.

Дан массив:

abc = ['A', 'B', 'C', 'D', 'E', 'F', 'G']

Оставить в нём только первый, предпоследний и последний элемент. Выполнить задание с помощью del.

Выходные данные должны иметь следующий вид:

>>> ['A', 'F', 'G']

Задание 3.

Дан список:

numbers = [1, 4]

Добавить в него недостающие цифры в порядке возрастания, используя метод insert().

Выходные данные должны иметь следующий вид:

>>> [1, 2, 3, 4]

Задание 4.

Дан числовой массив:

mass = [14, -6, 3, 11, 6, 8.5, 99, -20, -6, 10, 40, 0.25, -4, 5]

Удалить из него все отрицательные числа и отсортировать в порядке возрастания и убывания чисел.

Результат должен выглядеть следующим образом:

>>> [0.25, 3, 5, 6, 8.5, 10, 11, 14, 40, 99]

>>> [99, 40, 14, 11, 10, 8.5, 6, 5, 3, 0.25]

Задание 5

Даны 3 списка: для чисел с отрицательными значениями, для положительных чисел и для нулей.

Пользователь вводит сначала количество вводимых чисел, а затем сами числа.

Пример:

```
Введите количество чисел: 3
Введите числа: -1
Введите числа: 0
Введите числа: 5

Process finished with exit code 0
```

Далее, числа сортируются по соответствующим ими спискам.

Вывести сумму чисел списка с отрицательными числами и среднее арифметическое списка с положительными числами (при том, что если положительных чисел не будет, не будет ошибки деления на ноль).

А со списком, в котором находятся нули, сделать следующее:

1. Заменить нули знаком *
2. Вывести количество элементов списка и сами элементы.

Пример входных и выходных данных:

```
Введите количество чисел: 7
Введите числа: 0
Введите числа: 0
Введите числа: -2
Введите числа: -3
Введите числа: 2
Введите числа: 6
Введите числа: 0
-5
4.0
Количество звёзд: 3 ['*', '*', '*']
```

Критерии выставления оценки.

Оценка 5 – правильно решены и вовремя сданы 5 заданий. Все программы работают без ошибок и выполняются до конца.

Оценка 4 - правильно решены и вовремя сданы 4 задания. Все программы работают без ошибок и выполняются до конца.

Оценка 3 – написаны только 2 программы по заданиям. При этом алгоритм построен верно и программы работают.

Оценка 2 – выполнено меньше, чем 2 задания.