	и защиты современные компьютерные технологии
ОПК-5	способностью использовать основные приемы обработки и представления экспериментальных данных
ОПК-6	способность осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий

Теоретическая часть.

Большинство языков программирования могут представлять последовательность в виде объектов, проиндексированных с помощью их позиции, выраженной целым числом: первый, второй и далее до последнего. Вы уже знакомы со строками, они являются последовательностями символов. Вы уже немного знакомы со списками, они являются последовательностями, содержащими данные любого типа.

В Python есть еще две структуры-последовательности: кортежи и списки. Они могут содержать ноль или более элементов. В отличие от строк элементы могут быть разных типов. Фактически каждый элемент может быть любым объектом Python. Это позволяет создавать структуры любой сложности и глубины.

Почему же в Python имеются как списки, так и кортежи? Кортежи неизменяемы, когда вы присваиваете кортежу элемент, он «запекается» и больше не изменяется. Списки же можно изменять — добавлять и удалять элементы в любой удобный момент. Мы рассмотрим множество примеров применения обоих типов, сделав акцент на списках.

Списки

Списки служат для того, чтобы хранить объекты в определенном порядке, особенно если порядок или содержимое могут изменяться. В отличие от строк список можно изменить. Вы можете изменить список, добавить в него новые элементы, а также удалить или перезаписать существующие. Одно и то же значение может встречаться в списке несколько раз.

Кортежи

Кортежи, как и списки, являются последовательностями произвольных элементов. В отличие от списков кортежи неизменяемы. Это означает, что вы не можете добавить, удалить или изменить элементы кортежа после того, как определите его.

Поэтому кортеж аналогичен константному списку.

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

Создание списков с помощью оператора [] или метода list()

Список можно создать из нуля или более элементов, разделенных запятыми и заключенных в квадратные скобки:

```
>>> empty_list = []
>>> weekdays = ['Понедельник', 'Вторник', 'Среда', 'Четверг']
>>> big_birds = ['Эму', 'Орёл', 'Пингвин']
>>> first_names = ['Иван', 'Марья', 'Шелдон', 'Шелдон', 'Эми']
```

Кроме того, с помощью функции list() можно создать пустой список:

```
>>> another_empty_list = list()
>>> another_empty_list
[]
```

В данном примере только список weekdays использует тот факт, что элементы стоят в определенном порядке. Список first_names показывает, что значения не должны быть уникальными.

> Замечание:
>
> Если вы хотите размещать в последовательности только уникальные значения, множество (set) может оказаться лучшим вариантом, чем список. В предыдущем примере список big_birds вполне может быть множеством. О множествах вы сможете прочесть в следующей работе.

Преобразование других типов данных в списки с помощью функции list()

Функция list() преобразует другие типы данных в списки. В следующем примере строка преобразуется в список, состоящий из односимвольных строк:

```
>>> list('котэ')
['к', 'о', 'т', 'э']
```

В этом примере кортеж (этот тип мы рассмотрим сразу после списков) преобразуется в список:

```
>>> a_tuple = ('На старт', 'Внимание', 'Марш')
>>> list(a_tuple)
['На старт', 'Внимание', 'Марш']
```

Как мы делали в подразделе «Разделяем строку с помощью функции split()» лабораторной работы 2, можно использовать функцию split(), чтобы преобразовать строку в список, указав некую строку-разделитель:

```
>>> birthday = '1/6/1990'
>>> birthday.split('/')
['1', '6', '1990']
```

Что, если в оригинальной строке содержится несколько включений строки-разделителя подряд? В этом случае в качестве элемента списка вы получите пустую строку:

```
>>> a_str = 'a/b//c/d///e'
>>> a_str.split('/')
['a', 'b', '', 'c', 'd', '', '', 'e']
```

Если бы вы использовали разделитель //, состоящий из двух символов, то получили бы следующий результат:

```
>>> a_str.split('//')
['a/b', 'c/d', '/e']
```

Получение элемента с помощью конструкции [смещение]

Как и для строк, вы можете извлечь одно значение из списка, указав его смещение:

```
>>> friends = ['Рейчел', 'Моника', 'Фиби']
>>> friends[0]
'Рейчел'
>>> friends[1]
'Моника'
>>> friends[2]
'Фиби'
```

Опять же, как и в случае со строками, отрицательные индексы отсчитываются с конца строки:

```
>>> friends[-1]
'Фиби'
>>> friends[-2]
'Моника'
>>> friends[-3]
'Рейчел'
```

Замечание:

Смещение должно быть корректным значением для списка — оно представляет собой позицию, на которой располагается присвоенное ранее значение. Если вы укажете позицию, которая находится перед списком или после него, будет сгенерировано исключение (ошибка). Вот что случится, если мы попробуем получить шестого другafriends (смещение равно 5, если считать от нуля) или же пятого перед списком:

```
>>> friends = ['Чендлер', 'Росс', 'Джоуи']
>>> friends[5]
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    friends[5]
IndexError: list index out of range
>>> friends[-5]
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    friends[-5]
IndexError: list index out of range
```

Списки списков

Списки могут содержать элементы различных типов, включая другие списки, что показано далее:

```
>>> small_birds = ['Воробей', 'Синица']
>>> extinct_birds = ['Додо', 'Археоптерикс', 'Азиатский страус']
>>> flightless_birds = [3, 'Киви', 2, 'Эму']
>>> all_birds = [small_birds, extinct_birds, 'Волнистый попугай', flightless_birds]
```

Как же будет выглядеть список списков all_birds?

```
>>> all_birds
[['Воробей', 'Синица'], ['Додо', 'Археоптерикс', 'Азиатский страус'], 'Волнистый попугай',
[3, 'Киви', 2, 'Эму']]
```

Взглянем на его первый элемент:

```
>>> all_birds[0]
['Воробей', 'Синица']
```

Первый элемент является списком — это список small_birds, он указан как первый элемент списка all_birds. Вы можете догадаться, чем является второй элемент:

```
>>> all_birds[1]
['Додо', 'Археоптерикс', 'Азиатский страус']
```

Это второй указанный нами элемент, extinct_birds. Если нужно получить первый элемент списка extinct_birds, мы можем извлечь его из списка all_birds, указав два индекса:

```
>>> all_birds[1][0]
'Додо'
```

Индекс [1] ссылается на второй элемент списка all_birds, а [0] — на первый элемент внутреннего списка.

Изменение элемента с помощью конструкции [смещение]

По аналогии с получением значения списка с помощью его смещения вы можете изменить это значение:

```
>>> bros = ['Маршал', 'Барни', 'Тед']
>>> bros[0] = 'Робин Щербацки'
>>> bros
['Робин Щербацки', 'Барни', 'Тед']
```

Опять же смещение должно быть корректным для заданного списка.

Вы не можете изменить таким способом символ в строке, поскольку строки неизменяемы. Списки же можно изменить. Можете изменить количество элементов в списке, а также сами элементы.

Отрежьте кусочек — извлечение элементов с помощью диапазона смещений

Можно извлечь из списка подпоследовательность, использовав разделение списка:

```
>>> bros = ['Маршал', 'Барни', 'Тед']
>>> bros[1:3]
['Барни', 'Тед']
```

Такой фрагмент списка также является списком.

Как и в случае со строками, при разделении можно пропускать некоторые значения. В следующем примере мы извлечем каждый нечетный элемент:

```
>>> bros[::2]
['Маршал', 'Тед']
```

Теперь начнем с последнего элемента и будем смещаться влево на 2:

```
>>> bros[::-2]
['Тед', 'Маршал']
```

И наконец, рассмотрим прием инверсии списка:

```
>>> bros[::-1]
['Тед', 'Барни', 'Маршал']
```

Добавление элемента в конец списка с помощью метода append()

Традиционный способ добавления элементов в список — вызов метода append(), чтобы добавить их в конец списка. Добавим к ранее созданному списку друзей Рейчел, предварительно выведя его на экран:

```
>>> friends
['Чендлер', 'Росс', 'Джоуи']
>>> friends.append('Рейчел')
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Рейчел']
```

Объединяем списки с помощью метода extend() или оператора +=

Можно объединить один список с другим с помощью метода extend(). Предположим, что добрый человек дал нам новый список братьев Маркс, который называется others, и мы хотим добавить его в основной список marxes:

```
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Рейчел']
>>> others = ['Моника', 'Фиби']
>>> friends.extend(others)
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Рейчел', 'Моника', 'Фиби']
```

Можно также использовать оператор +=:

```
>>> friends+=others
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Рейчел', 'Моника', 'Фиби', 'Моника', 'Фиби']
```

Если бы мы использовали метод append(), список others был бы добавлен как один элемент списка, вместо того чтобы объединить его элементы со списком marxes:

```
>>> friends = ['Чендлер', 'Росс', 'Джоуи', 'Рейчел']
>>> others = ['Моника', 'Фиби']
>>> friends.append(others)
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Рейчел', ['Моника', 'Фиби']]
```

Это снова демонстрирует, что список может содержать элементы разных типов. В этом случае — четыре строки и список из двух строк.

Добавление элемента с помощью функции insert()

Функция append() добавляет элементы только в конец списка. Когда вам нужно добавить элемент в заданную позицию, используйте функцию insert(). Если вы укажете позицию 0, элемент будет добавлен в начало списка. Если позиция находится за пределами списка, элемент будет добавлен в конец списка, как и в случае с функцией append(), поэтому вам не нужно беспокоиться о том, что Python сгенерирует исключение:

```
>>> friends.insert(3,'Золушка')
>>> friends.insert(10,'Барри Аллен')
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Золушка', 'Рейчел', ['Моника', 'Фиби'],
'Барри Аллен']
```

Удаление заданного элемента с помощью функции del

Наши консультанты только что проинформировали нас о том, что Барри Аллен не был одним из друзей. Отменим последний ввод:

```
>>> del friends[-1]
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Золушка', 'Рейчел', ['Моника', 'Фиби']]
```

Когда вы удаляете заданный элемент, все остальные элементы, которые идут следом за ним, смещаются, чтобы занять место удаленного элемента, а длина списка уменьшается на единицу. Если вы удалите 'Золушка' из последней версии списка, то получите такой результат:

```
>>> friends[3]
'Золушка'
>>> del friends[3]
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Рейчел', ['Моника', 'Фиби']]
```

> **Примечение:**
>
> *del является оператором Python, а не методом списка — нельзя написать friends[-2].del(). Он похож на противоположную присваиванию (=) операцию: открепляет имя от объекта Python и может освободить память объекта, если это имя являлось последней ссылкой на нее.*

Удаление элемента по значению с помощью функции remove()

Если вы не знаете точно или вам все равно, в какой позиции находится элемент, используйте функцию remove(), чтобы удалить его по значению. Прощай, Моника:

```
>>> friends = ['Чендлер', 'Росс', 'Джоуи', 'Рейчел', 'Моника', 'Фиби']
>>> friends.remove('Моника')
>>> friends
['Чендлер', 'Росс', 'Джоуи', 'Рейчел', 'Фиби']
```

Получение заданного элемента и его удаление с помощью функции pop()

Вы можете получить элемент из списка и в то же время удалить его с помощью функции pop(). Если вызовете функцию pop() и укажете некоторое смещение, она возвратит элемент, находящийся в заданной позиции; если аргумент не указан, будет использовано значение –1. Так, вызов pop(0) вернет головной (начальный) элемент списка, а вызов pop() или pop(-1) — хвостовой (конечный) элемент, как показано далее:

```
>>> friends = ['Чендлер', 'Росс', 'Джоуи', 'Рейчел']
>>> friends.pop(-1)
'Рейчел'
>>> friends
['Чендлер', 'Росс', 'Джоуи']
```

> ***Примечание:***
>
> *Если вы используете функцию append(), чтобы добавить новые элементы в конец списка, и функцию pop(), чтобы удалить из конца этого же списка, вы реализуете структуру данных, известную как LIFO (last in, Frst out — «последним пришел — первым ушел»). Такую структуру чаще называют стеком. Вызов pop(0) создаст очередь FIFO (Frst in Frst out — «первым пришел — первым ушел»). Эти структуры могут оказаться полезными, если вы хотите собирать данные по мере их поступления и работать либо с самыми старыми (FIFO), либо с самыми новыми (LIFO).*

Определение смещения элемента по значению с помощью функции index()

Если вы хотите узнать смещение элемента в списке по его значению, используйте функцию index():

```
>>> wizards = ['Гарри', 'Рон', 'Гермиона', 'Добби']
>>> wizards.index('Рон')
1
```

Проверка на наличие элемента в списке с помощью оператора in

В Python наличие элемента в списке проверяется с помощью оператора in:

```
>>> 'Добби' in wizards
True
>>> 'Петунья' in wizards
False
```

Одно и то же значение может встретиться больше одного раза. До тех пор пока оно находится в списке хотя бы в единственном экземпляре, оператор in будет возвращать значение True:

```
>>> words = ['to be', 'or not', 'to be']
>>> 'to be' in words
True
```

> *Примечание:*
>
> *Если вы часто проверяете наличие элемента в списке и вас не волнует порядок элементов, то для хранения и поиска уникальных значений гораздо лучше подойдет множество. О множествах мы поговорим чуть позже в этой работе.*

Определяем количество включений значения с помощью функции count()

Чтобы определить, сколько раз какое-либо значение встречается в списке, используйте функцию count():

```
>>> words
['to be', 'or not', 'to be']
>>> words.count('to be')
2
>>> words.count('question')
0
```

Преобразование списка в строку с помощью функции join()

В подразделе «Объединяем строки с помощью функции join()» прошлой работы функция join() рассматривается более подробно, но взгляните еще на один пример того, что можно сделать с ее помощью:

```
>>> wizards
['Гарри', 'Рон', 'Гермиона', 'Добби']
>>> ','.join(wizards)
'Гарри,Рон,Гермиона,Добби'
```

Но погодите, вам может показаться, что нужно делать все наоборот. Функция join() предназначается для строк, а не для списков. Вы не можете написать wizards.join(', '), несмотря на то что интуитивно это кажется правильным. Аргументом для функции join() является эта строка или любая итерабельная последовательность строк, включая список, и она возвращает строку. Если бы функция join() была только методом списка, вы не смогли бы использовать ее для других итерабельных объектов вроде кортежей и строк. Если вы хотите, чтобы она работала с любым итерабельным типом, нужно написать особый код для каждого типа, чтобы обработать объединение. Будет полезно запомнить: join() противоположна split(), как показано здесь:

```
>>> separator = '*'
>>> joined = separator.join(friends)
>>> joined
'Чендлер*Росс*Джоуи'
>>> separated = joined.split(separator)
>>> separated
['Чендлер', 'Росс', 'Джоуи']
>>> separated == friends
True
```

Меняем порядок элементов с помощью функции sort()

Вам часто нужно будет изменять порядок элементов по их значениям, а не по смещениям. Для этого Python предоставляет две функции:

- функцию списка sort(), которая сортирует сам список;
- общую функцию sorted(), которая возвращает отсортированную копию списка.

Если элементы списка являются числами, они по умолчанию сортируются по возрастанию. Если они являются строками, то сортируются в алфавитном порядке:

```
>>> friends
['Чендлер', 'Росс', 'Джоуи']
>>> sorted_friends = sorted(friends)
>>> sorted_friends
['Джоуи', 'Росс', 'Чендлер']
```

sorted_friends — это копия, ее создание не изменило оригинальный список:

```
>>> friends
['Чендлер', 'Росс', 'Джоуи']
```

Но вызов функции списка sort() для friends изменит этот список:

```
>>> friends.sort()
>>> friends
['Джоуи', 'Росс', 'Чендлер']
```

Если все элементы вашего списка одного типа (в списке friends находятся только строки), функция sort() отработает корректно. Иногда можно даже смешать типы — например, целые числа и числа с плавающей точкой, — поскольку в рамках выражений они конвертируются автоматически:

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort()
>>> numbers
[1, 2, 3, 4.0]
```

По умолчанию список сортируется по возрастанию, но вы можете добавить аргумент reverse=True, чтобы отсортировать список по убыванию:

```
>>> numbers = [2, 1, 4.0, 3]
>>> numbers.sort(reverse=True)
>>> numbers
[4.0, 3, 2, 1]
```

Получение длины списка с помощью функции len()

Функция len() возвращает количество элементов списка:

```
>>> len(friends)
3
```

Присваивание с помощью оператора =, копирование с помощью функции copy()

Если вы присваиваете один список более чем одной переменной, изменение списка в одном месте повлечет за собой его изменение в остальных, как показано далее:

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]
>>> b = a
>>> b
[1, 2, 3]
>>> a[0] = 'Оп'
>>> a
['Оп', 2, 3]
```

Что же находится в переменной b? Ее значение все еще равно [1, 2, 3] или изменилось на ['Оп', 2, 3]? Проверим:

```
>>> b
['Оп', 2, 3]
```

Переменная b просто ссылается на тот же список объектов, что и a, поэтому, независимо от того, с помощью какого имени мы изменяем содержимое списка, изменение отразится на обеих переменных:

```
>>> b
['Оп', 2, 3]
>>> b[0] = 'Ничосе!'
>>> b
['Ничосе!', 2, 3]
>>> a
['Ничосе!', 2, 3]
```

Вы можете скопировать значения в независимый новый список с помощью одного из следующих методов:

- функции copy();
- функции преобразования list();
- разбиения списка [:].

Оригинальный список снова будет присвоен переменной a. Мы создадим b с помощью функции списка copy(), c — с помощью функции преобразования list(), а d — с помощью разбиения списка:

```
>>> a = [1,2,3]
>>> b = a.copy()
>>> c = list(a)
>>> d = a[:]
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> d
[1, 2, 3]
```

Здесь списки b, c и d являются копиями a — это новые объекты, имеющие свои значения, не связанные с оригинальным списком объектов [1, 2, 3], на который ссылается a. Изменение a не повлияет на копии b, c и d:

```
>>> a[0] = 'Единица'
>>> a
['Единица', 2, 3]
>>> b
[1, 2, 3]
>>> c
[1, 2, 3]
>>> d
[1, 2, 3]
```

Кортежи. Создание кортежей с помощью оператора ()

Синтаксис создания кортежей несколько необычен, как мы увидим в следующих примерах.

Начнем с создания пустого кортежа с помощью оператора ():

```
>>> empty_tuple = ()
>>> empty_tuple
()
```

Чтобы создать кортеж, содержащий один элемент или более, ставьте после каждого элемента запятую. Это вариант для кортежей с одним элементом:

```
>>> one_physicist = 'Шелдон Купер',
>>> one_physicist
('Шелдон Купер',)
```

Если в вашем кортеже более одного элемента, ставьте запятую после каждого из них, кроме последнего:

```
>>> physicsts_tuple = 'Шелдон', 'Леонард', 'Радж'
>>> physicsts_tuple
('Шелдон', 'Леонард', 'Радж')
```

При отображении кортежа Python выводит на экран скобки. Вам они совсем не нужны — кортеж определяется запятыми, — однако не повредят. Вы можете окружить ими значения, что позволяет сделать кортежи более заметными:

```
>>> physicsts_tuple = ('Шелдон', 'Леонард', 'Радж')
>>> physicsts_tuple
('Шелдон', 'Леонард', 'Радж')
```

Кортежи позволяют вам присвоить несколько переменных за один раз:

```
>>> a,b,c = physicsts_tuple
>>> a
'Шелдон'
>>> b
'Леонард'
>>> c
'Радж'
```

Иногда это называется распаковкой кортежа.

Вы можете использовать кортежи, чтобы обменять значения с помощью одного выражения, без применения временной переменной:

```
>>> password = 'Печеньки'
>>> dessert = 'Мороженое'
>>> password, dessert = dessert, password
>>> password
'Мороженое'
>>> dessert
'Печеньки'
```

Функция преобразования tuple() создает кортежи из других объектов:

```
>>> physicsts_list = ['Шелдон', 'Крипке', 'Лесли Винкл']
>>> tuple(physicsts_list)
('Шелдон', 'Крипке', 'Лесли Винкл')
```

Кортежи против списков

Вы можете использовать кортежи вместо списков, но они имеют меньше возможностей — у них нет функций append(), insert() и т. д., поскольку кортеж не может быть изменен после создания. Почему же не применять везде списки вместо кортежей?

- Кортежи занимают меньше места.
- Вы не сможете уничтожить элементы кортежа по ошибке.
- Вы можете использовать кортежи как ключи словаря (см. следующую работу).
- Именованные кортежи могут служить более простой альтернативой объектам.
- Аргументы функции передаются как кортежи.

При решении повседневных задач вы будете чаще использовать списки и словари, которые будут рассмотрены в следующей работе.

Контрольные вопросы

1. В чём отличие списков от строк?
2. Какие способы создания списков вы знаете?
3. Присвойте переменной ***a***строковое значение «котэ», затем переменной b присвойте значение - список из букв переменной ***a***. Затем в переменной ***c*** создайте из списка ***b*** строку, равную ***a***(используйте для этого команду из лабораторной работы по теме «Строки»).
4. Какая функция получает из строки список? Как задать разделитель, который должен быть использован?
5. Что будет, если при создании списка из строки разделитель встретился несколько раз подряд?
6. Как создать список списков? Как выводятся его элементы? Приведите пример.
7. Можно ли заменить элемент списка?
8. Как вывести диапазон элементов списка?

9. Как вывести на экран каждый второй элемент списка от последнего к первому?
10. Как вывести элементы списка на экран от последнего до первого?
11. Какие способы добавления элементов в список вы знаете?
12. Какие способы удаления элементов из списка вы знаете? В чём их отличие?
13. Как объединить два списка?
14. Что такое методы FIFOи LIFO?
15. Как определить номер элемента в списке по значению?
16. Как можно определить наличие элемента в списке?
17. Как можно определить количество вхождений элемента в список?
18. Как преобразовать список в строку?
19. В чём разница между функциями sortи sorted?
20. Каков порядок сортировки списка по умолчанию?
21. Как отсортировать список в обратном порядке?
22. Как определить длину списка?
23. В чём особенность присвоения значения одной переменной, содержащей список, другой?
24. Как можно скопировать список, сделав его независимым?
25. В чём состоит особенность кортежей?
26. Как создать кортеж? Какие способы вы знаете?
27. Какие достоинства и недостатки кортежей по сравнению со списками вы знаете?

Список литературы, рекомендуемый к использованию по данной теме:

1. Орлов, С. А. Программная инженерия. Технологии разработки программного обеспечения : учебник / С.А. Орлов. - 5-е изд., обновл. и доп. - СПб. : Питер, 2017. - 640 с.
2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники : [учеб. пособие] / Л.Г. Гагарина, А.А. Петров. - М. : Форум, 2016. - 368 с.
3. Шкляр, М. Ф. Основы научных исследований : учеб. пособие / М.Ф. Шкляр. - 6-е изд. - М. : Дашков и Ко, 2016. - 208 с.
4. Михеева, Е. В. Информационные технологии в профессиональной деятельности : учеб. пособие / Е.В. Михеева. - 14-е изд., стер. - М. : Академия, 2016. - 384 с.
5. Любанович Билл. Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.

Лабораторная работа 4. Словари, множества в языке Python.

Цель работы:

Изучить особенности структур словарей и множеств, основные функции для работы с ними, особенности и области их применения.

Компетенции:

Код	Формулировка:
ОК-5	быть готовым работать с информацией в различных формах, использовать для ее получения, обработки, передачи, хранения и защиты современные компьютерные технологии