МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

# Методические указания

по выполнению лабораторных работ
по дисциплине
**«ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ»**
для направления подготовки **10.03.01 Информационная безопасность**
направленность (профиль) **Безопасность компьютерных систем**

Пятигорск, 2024

## 1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью изучения дисциплины «Программирование на языках высокого уровня» является ознакомление учащихся с основами современных информационных технологий, тенденциями их развития, получение устойчивых навыков самостоятельной работы на персональном компьютере с применением современных программных средств для получения, хранения и обработки информации, а также получение навыков самостоятельного освоения новых программных средств.

В соответствии с указанной целью при изучении дисциплины «Программирование на языках высокого уровня» ставятся следующие задачи:

- дать общие характеристики процессов сбора, передачи, обработки и накопления информации;
- познакомить с основами кодирования и сжатия информации;
- дать сведения о технических и программных средствах реализации информационных процессов;
- ознакомить с современными операционными системами и оболочками;
- дать принципы организации, структуры средств систем мультимедиа и компьютерной графики;
- привить навыки работы на современном ПК.

## 2. НАИМЕНОВАНИЕ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

### Лабораторная работа 1.Введение в язык Python. Целочисленная арифметика. Типы данных

***Цель работы:***

Изучить особенности и области применения языка Python, научиться устанавливать интерпретатор языка и интегрированную среду разработки.Изучить интерфейс и работу с интерпретатором и IDLE. Изучить способы задания значений переменных и задания (изменения) их типов.

**Теоретическая часть.**

**Python** — современный объектно-ориентированный язык программирования, имеющий интерпретаторы для всех распространенных платформ. Язык программирования Питон разрабатывается чуть более 20 лет. В настоящее время распространены две версии языка — вторая и болееактуальная 3 версия. Питон версии 2 не развивается, но до сих пор широко используется, поскольку большое количество программного обеспечения и библиотек разработано именно для этой версии языка. Между версиями есть ряд синтаксических отличий, приводящих к несовместимости исходного кода приложений, в том числе в синтаксисе команд ввода-вывода, но в целом они очень похожи. Мы будет использовать именно версию 3, как более современную и актуальную.

**Python** — современный универсальный интерпретируемый язык программирования. Его достоинства:

1. Кроссплатформенность и бесплатность.
2. Простой синтаксис и богатые возможности позволяют записывать программы очень кратко, но в то же время понятно.
3. По простоте освоения язык сравним с бейсиком, но куда более богат возможностями и значительно более современен.
4. Богатая стандартная библиотека, возможность разработки промышленных приложений (для работы с сетью, GUI, базами данных и т.д.)

5. Огромное количество сторонних библиотек для решения практически любых задач в любой предметной области.

В настоящее время язык активно набирает популярность как универсальный язык программирования. Большинство межвузовских олимпиад по информатике предлагают к использованию язык Python. С 2015 года в текстах задач ЕГЭ примеры приводятся также на этом языке. Благодаря большому количеству готовых библиотек и фреймворков, появилась возможность применять Python как для научных вычислений, работы с BigData, нейросетями, так и для системного администрирования и даже для Web-разработки.

***Переменные в Python***

Переменная — это простейшая именованная структура данных, в которой может быть сохранён промежуточный или конечный результат работы программы.

Переменную в Python создать очень просто — нужно присвоить некоторому идентификатору значение при помощи оператора присваивания «=».

ПРИМЕР

```
a = 10
b = 3.1415926
c = «Hello»
d = [1, 2, 3]
```

В этом примере используются четыре переменные:

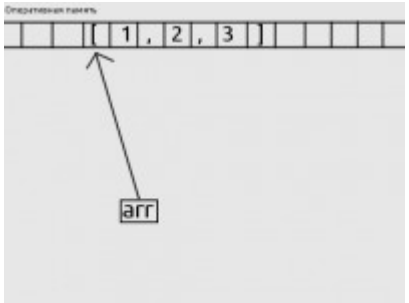
- переменная **a** хранит значение типа **int** (целое число),
- переменная **b** — типа **float** (действительное число),
- переменная **c** — типа **str** (строка),
- переменная **d** — типа **list** (список, в данном случае из трех целых чисел).

Никакого специального объявления переменных не требуется, первое присваивание переменной значения и является ее объявлением. Идентификатор в Python является «ссылкой» на хранимые в памяти данные. Python — язык с динамической типизацией: каждая переменная в каждый момент времени имеет определенный тип, но этот тип может меняться по ходу выполнения программы, достаточно просто присвоить ей новое значение другого типа.

На самом деле переменная в python является лишь ссылкой на объект в памяти. При создании любой переменной (число, строка или массив) в неё записывается ссылка на объект, а сам объект находится где-то в оперативной памяти далеко от самой переменной со ссылкой. Таким образом, несколько переменных могут указывать на один объект, и при изменении объекта (например, списка) изменится результат обращения к нему с использованием каждой переменной.

Происходящее при выполнении следующего кода схематично можно изобразить так:

arr = [1, 2, 3]




Имена переменных могут содержать только следующие символы:

- буквы в нижнем регистре (от «a» до «z»);
- буквы в верхнем регистре (от «A» до «Z»);

- цифры (от 0 до 9);
- нижнее подчеркивание (_).

Имена не могут начинаться с цифры. Python также особо обрабатывает имена, которые начинаются с нижнего подчеркивания (они будут рассмотрены в следующих работах). Корректными являются следующие имена:

- a;
- a1;
- a_b_c___95;
- _abc;
- _1a.

Следующие имена, однако, некорректны:

- 1;
- 1a;
- 1_.

Наконец, не следует использовать следующие слова для имен переменных, поскольку они являются зарезервированными словами Python:

| | | | | |
|---|---|---|---|---|
| false | class | finally | is | return |
| none | continue | for | lambda | try |
| true | def | from | nonlocal | while |
| and | del | global | not | with |
| as | elif | if | or | yield |
| assert | else | import | pass | |
| break | except | in | raise | |

Python имеет встроенную поддержку целых чисел (наподобие 5 и 1 000 000 000) и чисел с плавающей точкой (вроде 3,1416, 14,99 и 1,87e4). Вы можете вычислять комбинации чисел с помощью простых математических операторов, приведенных в таблице.

| **Оператор** | **Описание** | **Пример** | **Результат** |
|---|---|---|---|
| + | Сложение | 5 + 8 | 13 |
| – | Вычитание | 90 – 10 | 80 |
| * | Умножение | 4 * 7 | 28 |
| / | Деление с плавающей точкой | 7/2 | 3,5 |
| // | Целочисленное (Truncating) деление | 7//2 | 3 |
| % | Modulus (вычисление остатка) | 7%3 | 1 |
| ** | Возведение в степень | $3^4$ | 81 |

**Установка Python и сред разработки**

Для работы необходимо установить интерпретатор языка Питон и, для удобства написания программ, среду разработки.

***Установка интерпретатора***

Дистрибутивы для установки:

- Получить последнюю версию интерпретатора Python для вашей платформы можно на официальном сайте https://www.python.org/. В разделе Download можно выбрать установочные пакеты для Windows и MacOS.
- В операционных системахGNU/Linux язык Питон включен в репозитории всех популярных дистрибутивов. Используйте менеджер пакетов вашей ОС для

установки пакета python3. Возможно, он уже установлен в вашей системе: проверить это можно командой “python3” в терминале.

- Для Android есть пакет QPython3, который можно загрузить из PlayMarket

***Интегрированная среда разработки***

Для удобства разработки кода программисты используют среды разработки (IDE). Среда разработки помогает автоматизировать рутинные процессы разработки, такие как набор программного кода, запуск и отладка написанного проекта. В дистрибутив Python для Windows включена простая среда разработки IDLE.

Среда разработки состоит из консоли (PythonShell) позволяющей непосредственно выполнять операторы языка и простого текстового редактора с подсветкой синтаксиса общий вид которого приведен на Рисунке 1.

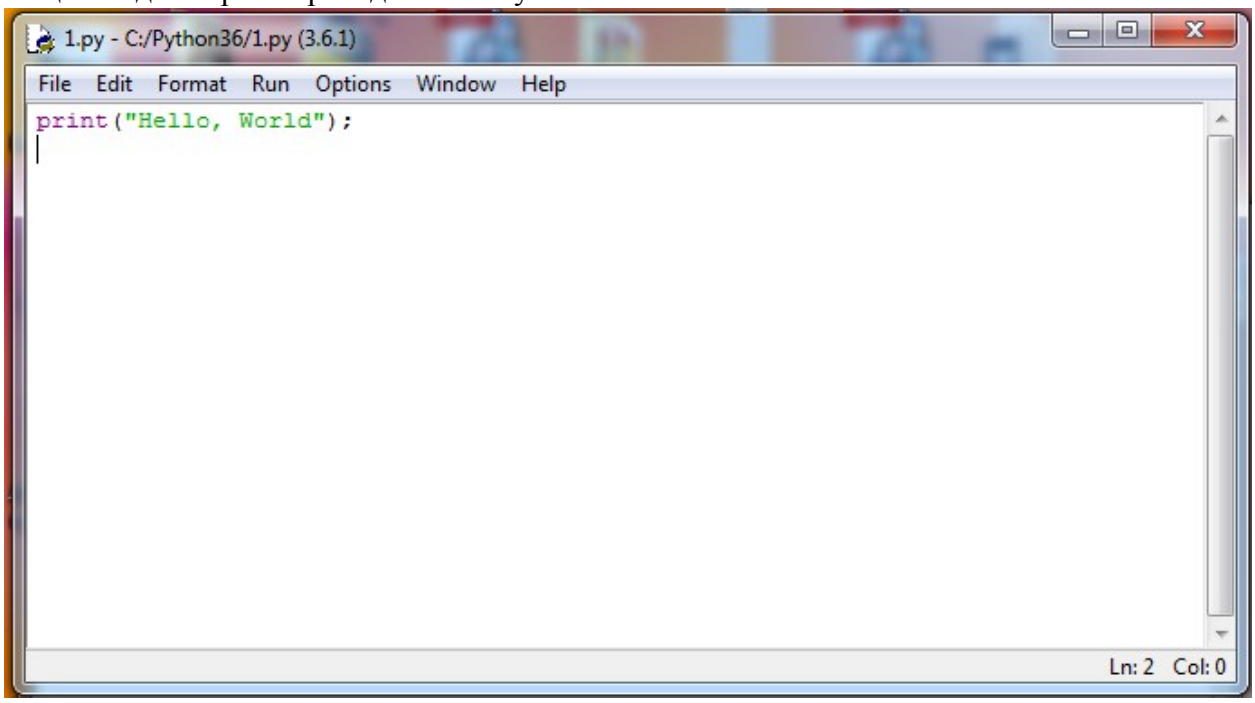



Рисунок 1 – главное окно редактора IDLE.

Рассмотрим основные элементы интерфейса редактора IDLE которые понадобятся для выполнения работ.

Меню File позволяет создать (New), сохранить (Save) и сохранить как (Saveas…) текущий файл на языке Python. Файл с исходными текстами на языке Python обычно имеет расширение ру. Пункты Close и Exit закрывают текущий файл и завершают работу IDLE соответственно.

Меню Edit содержит стандартные функции поиска, копирования и вставки.

Элементы меню Run используется для проверки и запуска написанной программы. Python Shell – открывает окно интерактивного интерпретатора Python. CheckModule – проверка синтаксиса кода в окне редактора. Перед проверкой необходимо сохранить изменения в файле (File - Save). В случае обнаружения синтаксической ошибки, редактор отобразит ее место и описание. Run Module – выполнение программы. Вначале выполняется проверка синтаксиса (CheckModule) и в том случае если ошибок не обнаружено, то запускается интерактивная оболочка PythonShell и выполняется написанная программа. Пример работы программы приведен на рисунке 2.

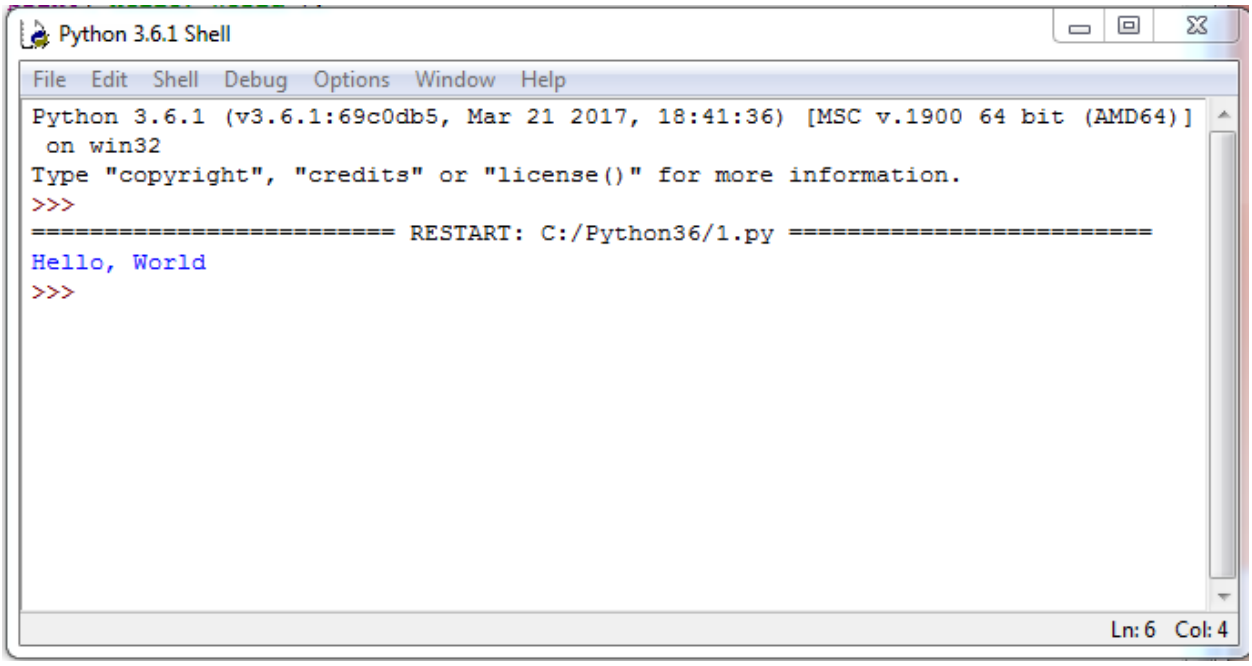



Рисунок 2 - Пример выполнения программы “Hello, World”

Интерактивный интерпретатор позволяет выполнять конструкции языка без создания файлов программ. Это удобный инструмент для экспериментирования с конструкциями языка и создания набросков алгоритмов. Пример работы со списками приведен на рисунке 3.

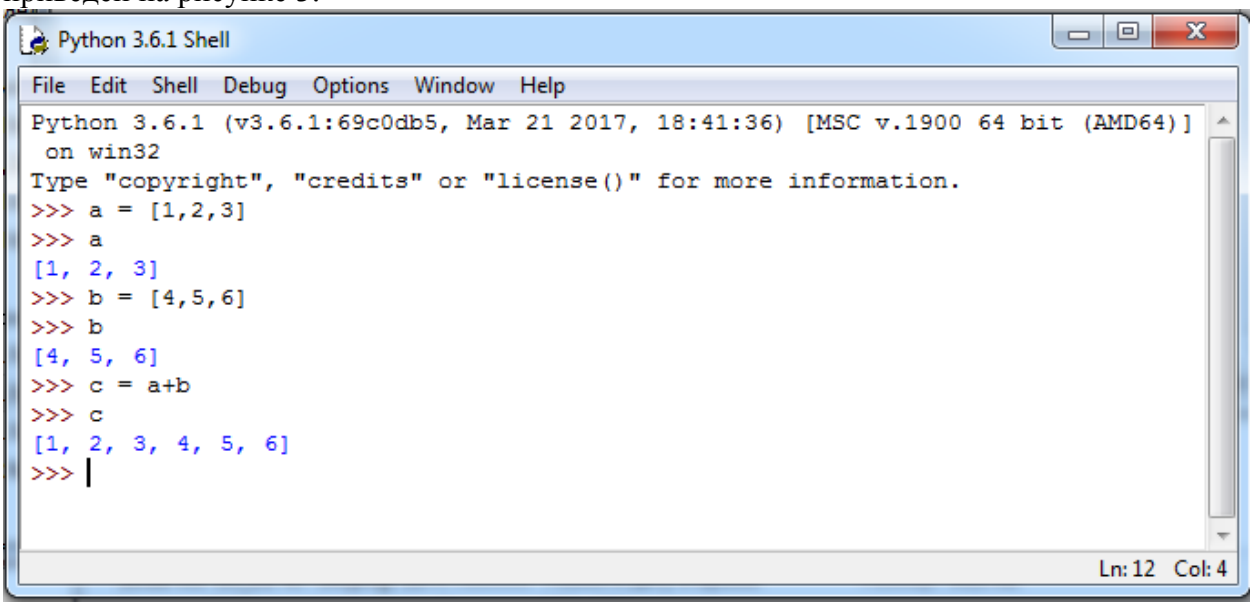



Рисунок – Интерактивный интерпретатор PythonShell. Пример работы со списками.

На рисунке 3 показано создание двух списков a и b, и объединение их в один новый список c. Для просмотра значения переменной, dPythonShell, достаточно написать ее имя и интерпретатор выведет ее значение.

Для больших проектов рекомендуется PyCharmCommunityEdition. Это бесплатная среда разработки, включающая все процессы связанные с разработкой, контролем версий, отладкой ПО, обновлением модулей и многим другим. Получить ее можно на официальном сайте разработчика https://www.jetbrains.com/pycharm/. Следует отметить, что существует так же образовательная версия PyCharmEdu содержащая примеры кода для всех основных языковых конструкций языка, получить ее можно бесплатно https://www.jetbrains.com/pycharm-edu/.

**Оборудование и материалы.**

Персональный компьютер, среда разработки Python.

**Указания по технике безопасности:**

Соответствуют технике безопасности по работе с компьютерной техникой.

**Задания**

Языки программирования также позволяют вам определять переменные. Переменные являются именами, которые ссылаются на значения в памяти компьютера. Вы можете определить их для использования в своей программе. В Python символ = применяется для присваивания значения переменной.

В следующей программе целое число 7 присваивается переменной с именем a, затем на экран выводится значение, связанное в текущий момент с этой переменной:

```
>>> a = 7
>>> print(a)
7
```

Сделайте следующее с помощью интерактивного интерпретатора.

1. Как и раньше, присвойте значение 7 имени a. Это создаст объект-«переменную», содержащую целочисленное значение 7.
2. Выведите на экран значение a.
3. Присвойте a переменной b, заставив b прикрепиться к объекту-«переменной», содержащей значение 7.
4. Выведите значение b.

```
>>> a = 7
>>> print(a)
7
>>> b = a
>>> print(b)
7
```

В Python, если вы хотите узнать тип какого-то объекта (переменной или значения), вам следует использовать конструкцию type(объект). Попробуем сделать это для разных значений (58, 99.9, abc) и переменных (a, b):

```
>>> type(a)
<class 'int'>
>>> type(b)
<class 'int'>
>>> type(99.9)
<class 'float'>
>>> type('abc')
<class 'str'>
```

***Целые числа***

Любая последовательность цифр в Python считается целым числом:

```
>>> 5
5
```

Последовательность цифр указывает на целое число. Если вы поместите знак + перед цифрами, число останется прежним, знак – сделает число отрицательным:

```
>>> 123
123
>>> +123
123
>>> -123
-123
```

С помощью Python вы можете выполнять обычные арифметические действия, как и с обычным калькулятором, используя операторы, показанные в теоретической части работы. Сложение и вычитание будут работать следующим образом:

```
>>> 5+9
14
>>> 100-7
93
>>> 4-10
-6
>>> 5+9+3
17
>>> 4 + 3 - 2 - 1 + 6
10
>>> 5+9  +      3
17
```

Вы можете заметить, что число пробелов в формуле не имеет значения.

Аналогично выглядит операция умножения.

Операция деления чуть более интересна, поскольку существует два ее вида:

- с помощью оператора / выполняется деление с плавающей точкой (десятичное деление);
- с помощью оператора // выполняется целочисленное деление (деление с остатком).

Даже если вы делите целое число на целое число, оператор / даст результат с плавающей точкой:

```
>>> 9/5
1.8
>>> 9//5
1
```

Деление на ноль с помощью любого оператора сгенерирует исключение:

```
>>> 5/0
Traceback (most recent call last):
  File "<pyshell#51>", line 1, in <module>
    5/0
ZeroDivisionError: division by zero
>>> 7//0
Traceback (most recent call last):
  File "<pyshell#52>", line 1, in <module>
    7//0
ZeroDivisionError: integer division or modulo by zero
```

Введите следующие команды:

```
>>> a = 95
>>> a
95
>>> a-3
92
```

Ранее, когда мы выполнили операцию a – 3, мы не присвоили результат переменной a, поэтому ее значение не изменилось. Выведите значение a на экран:

```
>>> print(a)
95
```

Если вы хотите изменить значение переменной a, придется сделать следующее:

```
>>> a = a - 3
>>> print(a)
92
```

Вы можете совместить арифметические операторы с присваиванием, размещая оператор перед знаком =. В этом примере выражение a -= 3 аналогично выражению a = a– 3. Введите в командную строку:

```
>>> a = 95
>>> a-=3
>>> a
92
```

Это выражение аналогично выражению a = a + 8:

```
>>> a+=8
>>> a
100
```

Аналогично работает умножение:

```
>>> a*=2
>>> a
200
```

Здесь представлен пример деления с плавающей точкой, a = a / 3:

```
>>> a/=3
>>> a
66.66666666666667
```

Присвоим значение 13 переменной a, а затем попробуем использовать сокращенный вариант a = a // 4 (целочисленное деление):

```
>>> a = 13
>>> a//=4
>>> a
3
```

Символ % имеет несколько разных применений в Python. Когда он находится между двух чисел, с его помощью вычисляется остаток от деления первого числа на второе:

```
>>> 13 % 5
3
```

Вот так можно получить частное и остаток одновременно:

```
>>> divmod(13,5)
(2, 3)
```

Только что вы увидели кое-что новое: функцию с именем divmod, в которую передаются целые числа 9 и 5, возвращающую двухэлементный результат, называемый кортежем. Кортежи будут рассмотрены в следующих работах.

***Системы счисления.***

В Python вы можете выразить числа в трех системах счисления помимо десятичной:

- 0b или 0B для двоичной системы (основание 2);
- 0o или 0O для восьмеричной системы (основание 8);
- 0x или 0X для шестнадцатеричной системы (основание 16).

Интерпретатор выведет эти числа как десятичные. Попробуем воспользоваться каждой из систем счисления. Первой выберем десятичное число 10, которое означает «одна десятка и ноль единиц». Затем то же число в двоичной системе:

```
>>> 10
10
>>> 0b10
2
```

Затем попробуйте перевод из восьмеричной и шестнадцатеричной систем:

```
>>> 0o10
8
>>> 0xf
15
>>> 0xef
239
```

***Преобразования типов***

Для того чтобы изменить другие типы данных на целочисленный тип, следует использовать функцию int(). Она сохраняет целую часть числа и отбрасывает любой остаток.

Простейший тип данных в Python — булевы переменные, значениями этого типа могут быть только True или False. При преобразовании в целые числа они представляют собой значения 1 и 0:

```
>>> int(True)
1
>>> int(False)
0
```

Преобразование числа с плавающей точкой в целое число просто отсекает все, что находится после десятичной запятой:

```
>>> int(98.6)
98
>>> int(1.0e4)
10000
```

Далее рассмотрим пример преобразования текстовой строки (со строкамив Python вы познакомитесь в следующей лабораторной работе, но, возможно, знаете о них из других языков программирования), которая содержит только цифры и, возможно, знаки + и -:

```
>>> int('99')
99
>>> int('-23')
-23
>>> int('+12')
12
```

Применение операции int() к целому числуничего в нём не изменит.

Если вы попробуете преобразовать что-то непохожее на число, сгенерируется исключение:

```
>>> int('10 негритят пошли купаться в море')
Traceback (most recent call last):
  File "<pyshell#93>", line 1, in <module>
    int('10 негритят пошли купаться в море')
ValueError: invalid literal for int() with base 10: '10 негри
в море'
>>> int('')
Traceback (most recent call last):
  File "<pyshell#94>", line 1, in <module>
    int('')
ValueError: invalid literal for int() with base 10: ''
```

Текстовая строка в предыдущем примере начинается с корректных символов-цифр (10), но продолжается теми символами, которые функция int() обработать не может.

***Какова размерность целочисленного типа?***

В Python 2 размер переменной типа int был ограничен 32 битами. Этого было достаточно для того, чтобы сохранить любое целое число в диапазоне от –2 147 483 648 до 2 147 483 647. Для переменных типа long выделялось еще больше места — 64 бита, что позволяло хранить значения в диапазоне от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807.

В Python 3 тип long отсутствует, а переменная типа int может быть любого размера — даже больше 64 бит. Поэтому вы можете выполнить арифметические действия наподобие следующих (10**100 называется гугол (googol), компания Google называлась так до того, как решила сменить свое название на более простое):

```
>>> 10**100
10000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
0000000
```

***Вещественные числа***

Значениями целочисленного типа являются целые числа, но числа с плавающей точкой (в Python они называются float) имеют десятичную запятую. Числа с плавающей точкой обрабатываются так же, как и целые: вы можете использовать операторы +, –, *, /, //, **, % и функцию divmod().

Для того чтобы преобразовать другие типы в тип float, следует использовать

функцию float().

***Математические функции***

Python имеет привычный набор математических функций вроде квадратного корня, косинуса и т. д. Они будут рассмотрены в лабораторных работах, где будет рассмотрено применение Python в науке.

***Упражнения***

В этойработе были показаны атомы Python: числа, строки и переменные. Используя полученные знания, выполним несколько небольших упражнений по работе с ними с помощью интерактивного интерпретатора.

1. Сколько секунд в часе? Используйте интерактивный интерпретатор как калькулятор и умножьте количество секунд в минуте (60) на количество минут в часе (тоже 60).
2. Присвойте результат вычисления предыдущего задания (секунды в часе) переменной, которая называется seconds_per_hour.
3. Сколько секунд в сутках? Используйте переменную seconds_per_hour.
4. Снова посчитайте количество секунд в сутках, но на этот раз сохраните результат в переменной seconds_per_day.
5. Разделите значение переменной seconds_per_day на значение переменной seconds_per_hour. Используйте деление с плавающей точкой (/).
6. Разделите значение переменной seconds_per_day на значение переменной seconds_per_hour. Используйте целочисленное деление (//). Совпадает ли полученный результат с ответом на предыдущее упражнение, если не учитывать символы .0 в конце?

***Содержание отчета*:выполненную работу сохранить в папке под своей фамилией и показать преподавателю**

***Контрольные вопросы***

1. Каковы особенности синтаксиса языка Python?
2. Какие вычислительные операции применимы к целочисленным типам данных? К вещественным?
3. Что такое строковый тип? Какие операции применимы к строковому типу?
4. Каковы особенности ввода данных в Python?
5. Каковы особенности вывода данных в Python?
6. Что такое IDLE?
7. Какой командой определяется тип переменной?
8. Как применить операцию деления нацело?
9. Что произойдет при делении на 0 с помощью оператора /? С помощью //?
10. Если ввести последовательность команд

```
>>> a = 95
>>> a
95
>>> a – 3
92
```

Что будет выведено после команды print(a)?

11. Как теперь получить a = 92?
12. Что означает выражение a -= 3?
13. Какие способы вывода значения переменной на экран в IDLEвы знаете?
14. Что будет выведено в результате?

```
>>> a = 13
>>> a//=4
>>> a
```

15. Что получится в результате вычисления выражения:

```
>>> 13 % 5
3
```

16. Что это за операция? Что получается в результате?

```
>>> divmod(13,5)
(2, 3)
```

17. Что будет выведено в результате команды

```
>>> 0o10
```

18. Что будет выведено в результате

```
>>> 0xf
```

19. Что будет выведено в результате команды

```
>>> 0b10
```

20. Что будет выведено в результате команды

```
>>> 0b100
```

21. Что будет выведено в результате применения функции:

```
>>> int(5.2e5)
```

22. Что будет выведено в результате команды:

```
>>> True + 2
```

23. Что будет выведено в результате команды:

```
>>> 10**100
```

**Лабораторная работа 2. Работа со строками в языке Python.**

***Цель работы:***

Изучить способы создания строк и функции для работы с ними: слияние строк, изменение регистра, получение подстрок, выравнивание.

***Теоретическая часть.***

***Строковый тип данных и ввод-вывод строк***

В программировании один из наиболее часто используемых типов данных - строки. В строковых данных можно хранить фамилии и имена, адреса, названия товаров и т.д. Чтобы задать в программе на языке Python строку необходимо заключить последовательность символов в двойные или одинарные кавычки:

```
a="У лукоморья дуб зеленый"
b=''          # Это пустая строка
```

Считать строковую переменную с клавиатуры можно при помощи стандартной функции`input`, при этом пользователь также должен заключить вводимую строку в кавычки. Пример:

```
a=input()
```

***Операции со строками***

Главная операция со строками- их объединение, когда одна строка записывается после другой строки. Эта операция называется *конкатенацией* и для нее используется оператор`+`:

```
a="abc"
b="xyz"
c=a+b
print c       # Будет напечатано abcxyz
```

Также при помощи оператора `*` можно многократно повторять одну и ту же строку. "Умножать" строку можно только на натуральное число:

```
print "="*20  # Будет напечатано 20 знаков "=" подряд
```

В отличии от списков, изменять отдельные элементы элементы (то есть символы) в строке нельзя. Вместо этого можно пользоваться конкатенацией строк. Например, если хочется в строке `s` заменить первый символ на букву `"a"`, то это можно сделать при помощи конкатенации строки `"a"` и всей строки`s`за исключением первого символа: `s="a"+s[1:]`.

Рассмотрим пример программы, которая считывает строку с клавиатуры и находит в ней первое слово (то есть все символы до первого пробела.

```
s=input()              # 1
    for i in range(len(s)):      # 2
      if s[i]==" ":              # 3
        print s[:i]              # 4
break                  # 5
```

В первой строке программы в переменную `s` считывается строка с клавиатуры. Затем организовывается цикл, в котором переменная `s` меняется от 0 до `len(s)-1`, то есть переменная `i` принимает подряд все номера символов в строке. В третьей строке программы проверяется, является ли `i`-й символ строки пробелом (то есть совпадает ли он со строкой из одного пробела). Если проверяемое условие истинно, то был найден первый слева строки пробел, на экран печатается первые `i` символов строки, то есть все символы с начала строки до найденного пробела, после чего выполнение цикла завершается инструкцией `break`.

**Оборудование и материалы.**

Персональный компьютер, среда разработки Python.

**Указания по технике безопасности:**

Соответствуют технике безопасности по работе с компьютерной техникой.

## Задания

Благодаря поддержке стандарта Unicode Python 3 может содержать символы любого языка мира, а также многие другие символы. Необходимость работы с этим стандартом была одной из причин изменения Python 2.

Строки являются первым примером последовательностей в Python. В частности, они представляют собой последовательности символов.В отличие от других языков, в Python строки являются неизменяемыми. Вы не можете изменить саму строку, но можете скопировать части строк в другую строку, чтобы получить тот же эффект.

***Создаем строки с помощью кавычек***

Строка в Python создается заключением символов в одинарные или двойные кавычки, как показано в следующем примере:

```
>>> 'Печенюшки'
'Печенюшки'
>>> "Семки"
'Семки'
```

Интерактивный интерпретатор выводит на экран строки в одинарных кавычках, но все они обрабатываются одинаково. Зачем иметь два вида кавычек? Основная идея заключается в том, что вы можете создавать строки, содержащие кавычки. Внутри одинарных кавычек можно расположить двойные и наоборот.

Для задания строк можно использовать тройные кавычки, это удобно для создания многострочного блока текста:

```
>>> poem = '''Товарищ, верь, пройдёт она,
и демократия, и гласность.
И вот тогда госбезопасность
Припомнит наши имена!'''
```

(Это стихотворение было введено в интерактивный интерпретатор, который поприветствовал нас символами >>> в первой строке и выводил символы … до тех пор, пока мы не ввели последние тройные кавычки и не перешли к следующей строке.)

Если бы вы попробовали создать стихотворение с помощью одинарных кавычек, Python выдал бы ошибку, когда бы вы перешли к следующей строке.

Если внутри тройных кавычек располагается несколько строк, символы конца строки будут сохранены внутри нее. Если перед строкой или после нее находятся пробелы, они также будут сохранены.

Вам может понадобиться работать с пустой строкой. В ней нет символов, но она совершенно корректна. Вы можете создать пустую строку с помощью любых упомянутых ранее кавычек:

```
>>> ''
''
>>> ""
''
>>> ''''''
''
>>> """"""
''
```

Зачем может понадобиться пустая строка? Иногда приходится компоновать строку из других строк и для этого нужно начать с чистого листа, то есть с пустой строки.

```
>>> men = 15
>>> base = ''
>>> base+=str(men)
>>> base+=' человек на сундук мертвеца'
>>> base
'15 человек на сундук мертвеца'
```

***Преобразование типов данных с помощью функции str()***

Вы можете преобразовывать другие типы данных Python в строки с помощью функции str():

```
>>> str(98.6)
'98.6'
>>> str(1.0e4)
'10000.0'
>>> str(True)
'True'
```

***Создаем управляющие символы с помощью символа ***

Python позволяет вам создавать управляющие последовательности внутри строк, чтобы добиться эффекта, который по-другому было бы трудно выразить. Размещая перед символом обратный слеш (\), вы наделяете этот символ особым значением. Наиболее распространена последовательность \n, которая означает переход на новую строку. С ее помощью вы можете создать многострочные строки из однострочных. Наберите в командной строке EDLE:

```
>>> palindrome = 'Аргентина \nманит \nнегра'
>>> print(palindrome)
Аргентина
манит
негра
```

Вы также увидите последовательность \t (табуляция), которая используется для выравнивания текста:

```
>>> print('\tКрасная шапочка')
        Красная шапочка
>>> print('Красная \tшапочка')
Красная         шапочка
>>> print('Красная шапочка\t')
Красная шапочка
```

В последней строке табуляция стоит в конце, ее вы, конечно, увидеть не можете.

Если вам нужен обратный слеш, просто напечатайте два:

```
>>> speech = 'Сегодня нам понадобился обратный слеш: \\'
>>> print(speech)
Сегодня нам понадобился обратный слеш: \
```

***Объединяем строки с помощью символа +***

Вы можете объединить строки или строковые переменные в Python с помощью оператора +, как показано далее:

```
>>> 'Я обернулся посмотреть '+'не обернулась ли она'
'Я обернулся посмотреть не обернулась ли она'
```

Можно также объединять строки (не переменные), просто расположив одну перед другой:

```
>>> "чтоб посмотреть, " "не обернулся ли я"
'чтоб посмотреть, не обернулся ли я'
```

Не забывайте добавлять пробелы при объединении строк.

***Размножаем строки с помощью символа ****

Оператор * можно использовать для того, чтобы размножить строку. Попробуйте ввести в интерактивный интерпретатор следующие строки и посмотреть, что получится:

```
>>> start = 'Раз-два '*4 + '\n'
>>> end = 'Проверка связи.'
>>> print(start + end)
Раз-два Раз-два Раз-два Раз-два
Проверка связи.
```

***Извлекаем символ с помощью символов [ ]***

Для того чтобы получить один символ строки, задайте смещение внутри квадратных скобок после имени строки. Смещение первого (крайнего слева) символа равно 0, следующего — 1 и т. д. Смещение последнего (крайнего справа) символа может быть выражено как –1, поэтому вам не придется считать, в таком случае смещение последующих символов будет равно –2, –3 и т. д.:

```
>>> letters = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
>>> letters[0]
'а'
>>> letters[1]
'б'
>>> letters[-1]
'я'
>>> letters[-2]
'ю'
```

Если вы укажете смещение, равное длине строки или больше (помните, смещения лежат в диапазоне от 0 до длины строки –1), сгенерируется исключение:

```
>>> letters[33]
Traceback (most recent call last):
  File "<pyshell#137>", line 1, in <module>
    letters[33]
IndexError: string index out of range
```

Поскольку строки неизменяемы, вы не можете вставить символ непосредственно в строку или изменить символ по заданному индексу. Попробуем изменить слово Маша на слово Даша и посмотрим, что произойдет:

```
>>> name = 'Маша'
>>> name[0] = 'Д'
Traceback (most recent call last):
  File "<pyshell#139>", line 1, in <module>
    name[0] = 'Д'
TypeError: 'str' object does not support item assignment
```

Вместо этого вам придется использовать комбинацию строковых функций вроде replace() или slice (которая будет рассмотрена далее):

```
>>> name = 'Маша'
>>> name.replace('М','Д')
'Даша'
>>> 'Д' + name[1:]
'Даша'
```

***Извлекаем подстроки с помощью оператора [ start : end : step ]***

Из строки можно извлечь подстроку (часть строки) с помощью функции slice. Вы определяете slice с помощью квадратных скобок, смещения начала подстроки start и конца подстроки end, а также опционального размера шага step. Некоторые из этих параметров могут быть исключены. В подстроку будут включены символы, расположенные начиная с точки, на которую указывает смещение start, и заканчивая точкой, на которую указывает смещение end.

- Оператор [:] извлекает всю последовательность от начала до конца.
- Оператор [start :] извлекает последовательность с точки, на которую указывает смещение start, до конца.
- Оператор [: end] извлекает последовательность от начала до точки, на которую указывает смещение end минус 1.
- Оператор [start : end] извлекает последовательность с точки, на которую указывает смещение start, до точки, на которую указывает смещение end минус 1.

- Оператор [start : end : step] извлекает последовательность с точки, на которую указывает смещение start, до точки, на которую указывает смещение end минус 1, опуская символы, чье смещение внутри подстроки кратно step.

Как и ранее, смещение слева направо определяется как 0, 1 и т. д., а справа налево — как -1, -2 и т. д. Если вы не укажете смещение start, функция будет использовать в качестве его значения 0 (начало строки). Если вы не укажете смещение end, функция будет использовать конец строки.

Создадим строку, содержащую русские буквы в нижнем регистре:

```
>>> letters = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
```

Использование простого двоеточия аналогично использованию последовательности 0: (целая строка):

```
>>> letters[:]
'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
```

Вот так можно получить все символы, начиная с 20-го и заканчивая последним:

```
>>> letters[20:]
'уфхцчшщъыьэюя'
```

А теперь получим символы с 12-го по 14-й (Python не включает символ, расположенный под номером, который указан последним):

```
>>> letters[12:15]
'лмн'
```

Последние три символа:

```
>>> letters[-3:]
'эюя'
```

В следующем примере мы начинаем со смещения 18 и идем до четвертого с конца символа. Обратите внимание на разницу с предыдущим примером, где старт с позиции -3 получал символ э. В этом примере конец диапазона -3 означает, что последним будет символ по адресу -4 — ь:

```
>>> letters[18:-3]
'стуфхцчшщъыь'
```

В следующем примере мы получаем символы, начиная с шестого с конца и заканчивая третьим с конца:

```
>>> letters[-6:-2]
'ъыьэ'
```

Если вы хотите увеличить шаг, укажите его после второго двоеточия, как показано в нескольких следующих примерах.

Каждый седьмой символ с начала до конца:

```
>>> letters[::7]
'ажнфы'
```

Каждый третий символ, начиная со смещения 4 и заканчивая 19-м символом:

```
>>> letters[4:20:3]
'джймпт'
```

Каждый четвертый символ, начиная с 19-го:

```
>>> letters[19::4]
'тцъю'
```

Каждый пятый символ от начала до 20-го:

```
>>> letters[:21:5]
'аейоу'
```

Опять же значение end должно быть на единицу больше, чем реальное смещение.

И это еще не все! Если задать отрицательный шаг, Python будет двигаться в обратную сторону. В следующем примере движение начинается с конца и заканчивается в начале, ни один символ не пропущен:

```
>>> letters[-1::-1]
'яюэьыъщшчцхфутсрпонмлкйизжёедгвба'
```

Можно добиться того же результата, использовав такой пример:

```
>>> letters[::-1]
'яюэьыъщшчцхфутсрпонмлкйизжёедгвба'
```

Операция slice более мягко относится к неправильным смещениям, чем поиск по индексу. Если указать смещение меньшее, чем начало строки, оно будет обрабатываться как 0, а если указать смещение большее, чем конец строки, оно будет обработано как –1. Это показано в следующих примерах.

Начиная с –50-го символа и до конца:

```
>>> letters[-50:]
'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
```

Начиная с –51-го символа и заканчивая –50-м:

```
>>> letters[-51:-50]
''
```

***Получаем длину строки с помощью функции len()***

До этого момента мы использовали специальные знаки препинания вроде +, чтобы манипулировать строками. Но существует не так уж много подобных функций. Теперь мы начнем использовать некоторые встроенные функции Python: именованные фрагменты кода, которые выполняют определенные операции.

Функция len() подсчитывает символы в строке:

```
>>> len(letters)
33
>>> empty = ""
>>> len(empty)
0
```

Функция len() используется также для других структур данных, которые будут рассмотрены далее.

***Разделяем строку с помощью функции split()***

В отличие от функции len() некоторые функции характерны только для строк. Для того чтобы использовать строковую функцию, введите имя строки, точку, имя функции и аргументы, которые нужны функции: строка. функция(аргументы). Более подробно о функциях мы будем говорить позже.

Вы можете использовать встроенную функцию split(), чтобы разбить строку на список небольших строк, основываясь на разделителе. Со списками вы познакомитесь в следующей лабораторной работе. Список — это последовательность значений, разделенных запятыми и окруженных квадратными скобками:

```
>>> todos = 'купить молоко, помыть полы, почитать ребёнку'
>>> todos.split(',')
['купить молоко', ' помыть полы', ' почитать ребёнку']
```

В предыдущем примере строка имела имя todos, а строковая функция называлась split() и получала один аргумент ','. Если вы не укажете разделитель, функция split() будет использовать любую последовательность пробелов, а также символы новой строки и табуляцию:

```
>>> todos.split()
['купить', 'молоко,', 'помыть', 'полы,', 'почитать', 'ребёнку']
```

Если вы вызываете функцию split без аргументов, вам все равно нужно добавлять круглые скобки — именно так Python узнает, что вы вызываете функцию.

***Объединяем строки с помощью функции join()***

Функция join() является противоположностью функции split(): она объединяет список строк в одну строку. Вызов функции выглядит немного запутанно, поскольку сначала вы указываете строку, которая объединяет остальные, а затем — список строк для объединения: string.join(list). Для того чтобы объединить список строк lines, разделив их символами новой строки, вам нужно написать '\n'.join(lines). В следующем примере мы объединим несколько имен в список, разделенный запятыми и пробелами:

```
>>> cripto_list = ['Йети', 'Полтергейст', 'Лохнесское чудовище']
>>> cripto_string = ','.join(cripto_list)
>>> print('Они обитают в лесах Нечерноземья: ', cripto_string)
Они обитают в лесах Нечерноземья:  Йети,Полтергейст,Лохнесское чудовище
```

***Прочие функции для работы со строками***

Python содержит большой набор функций для работы со строками. Рассмотрим принцип работы самых распространенных из них. Объектом для тестов станет следующеестихотворение:

```
>>> poem = '''Был этот мир глубокой тьмой окутан.
Да будет свет! И вот явился Ньютон.
Но Сатана недолго ждал реванша -
Пришел Эйнштейн, и стало всё как раньше.'''
```

Для начала получим первые 13 символов (их смещения лежат в диапазоне от 0 до 12):

```
>>> poem[:13]
'Был этот мир '
```

Сколько символов содержит это стихотворение? (Пробелы и символы новой строки учитываются.)

```
>>> len(poem)
145
```

Начинается ли стихотворение с буквосочетания ***Был этот***?

```
>>> poem.startswith('Был этот')
True
```

Заканчивается ли оно буквосочетанием ***всё как***?

```
>>> poem.endswith('всё как')
False
```

Найдем первое вхождениеслова ***мир***:

```
>>> word = 'мир'
>>> poem.find(word)
9
```

Найдем последнее вхождение буквы ***и***:

```
>>> poem.rfind('и')
122
```

Сравните с первым:

```
>>> poem.find('и')
10
```

Сколько раз встречается буква ***и***?

```
>>> poem.count('и')
4
```

Являются ли все символы стихотворения буквами или цифрами?

```
>>> poem.isalnum()
False
```

Нет, в стихотворении имеются еще и знаки препинания.

***Регистр и выравнивание***

В этом разделе мы рассмотрим еще несколько примеров использования встроенных функций. В качестве подопытной выберем следующую строку:

```
>>> setup = 'американец, француз и русский попали на необитаемый остров...'
```

Удалим символ «.» с обоих концов строки:

```
>>> setup.strip('.')
'американец, француз и русский попали на необитаемый остров'
```

> Замечание:
>
> Поскольку строки неизменяемы, ни один из этих примеров не изменяет строку setup. Каждый пример просто берет значение переменной setup, выполняет над ним некоторое действие, а затем возвращает результат как

новую строку.

Напишем первое слово с большой буквы:

```
>>> setup.capitalize()
'Американец, француз и русский попали на необитаемый остров...'
```

Напишем все слова с большой буквы:

```
>>> setup.title()
'Американец, Француз И Русский Попали На Необитаемый Остров...'
```

Запишем все слова большими буквами:

```
>>> setup.upper()
'АМЕРИКАНЕЦ, ФРАНЦУЗ И РУССКИЙ ПОПАЛИ НА НЕОБИТАЕМЫЙ ОСТРОВ...'
```

Запишем все слова маленькими буквами:

```
>>> setup.lower()
'американец, француз и русский попали на необитаемый остров...'
```

Сменим регистры букв:

```
>>> setup.swapcase()
'АМЕРИКАНЕЦ, ФРАНЦУЗ И РУССКИЙ ПОПАЛИ НА НЕОБИТАЕМЫЙ ОСТРОВ...'
```

Если бы в строке были буквы в разных регистрах, они бы поменяли свой регистр на противоположный. В данном случае все буквы выведены в верхнем регистре.

Теперь мы поработаем с функциями выравнивания. Строка выравнивается внутри заданного количества пробелов (в данном примере 30). Отцентруем строку в промежутке из 30 пробелов:

```
>>> setup = 'Заходит лошадь в бар...'
>>> setup.center(30)
'   Заходит лошадь в бар...    '
```

Выровняем ее по левому краю:

```
>>> setup.ljust(30)
'Заходит лошадь в бар...       '
```

А теперь по правому:

```
>>> setup.rjust(30)
'       Заходит лошадь в бар...'
```

О форматировании и преобразовании строк мы более подробно поговорим далее. Также далее будет затронуто использование символа % и функции format().

***Заменяем символы с помощью функции replace()***

Вы можете использовать функцию replace() для того, чтобы заменить одну подстроку другой. Вы передаете в эту функцию старую подстроку, новую подстроку и количество включений старой подстроки, которое нужно заменить. Если вы опустите последний аргумент, будут заменены все включения. В этом примере совпадает с заданным значением и заменяется следующая строка:

```
>>> setup.replace('лошадь', 'панда')
'Заходит панда в бар...'
```

Заменим максимум 100 включений буквы ***а***:

```
>>> setup.replace('а', 'ааа', 100)
'Заааходит лошаааадь в бааар...'
```

Если вы точно знаете, какую подстроку или подстроки хотите изменить, функция replace() станет для вас хорошим выбором.

Иногда вам нужно убедиться, что подстрока является целым словом, началом слова и т. д. В этих случаях понадобятся регулярные выражения. Их мы рассмотрим в других работах.

***Содержание отчета :*** **выполненную работу сохранить в папке под своей фамилией и показать преподавателю**

***Контрольные вопросы***

1. В чём особенность строк по сравнению с другими составными типами данных в Python и со строками во многих других языках программирования?
2. Какие способы задания строк вы знаете?
3. Для чего используются два вида кавычек при задании строк?
4. Как можно соединить 2 строки в одну?
5. Что будет выведено после применения команды
```
>>> str(1.0e4)
```
6. Как извлечь из строки с русским алфавитом каждую вторую букву, начиная с «д»?
7. Какой командой можно вывести строку ‘Утро красит нежным цветом’ в обратном порядке?
8. Что будет выведено, если запросить срез переменной с русским алфавитом (33 буквы) следующим образом:
```
>>> letters[0:100]
```
9. Для чего предназначена функция len()?
10. Для чего предназначена функция split()?
11. Что будет результатом выполнения команд ?
```
>>> todos = 'купить молоко, помыть полы, почитать ребёнку'
>>> todos.split(',')
```
12. Что будет, если в предыдущем примере вызвать функцию split() без аргумента?
```
>>> todos.split()
```
13. Для чего предназначена функция join()? Приведите пример использования.
14. Для чего предназначена функция strip()?
15. Для чего предназначена функция capitalize? Что будет, если применить её к строке ‘переходи на тёмную сторону, юзернейм, у нас печеньки!’
16. Как изменится строка после применения функции capitalize?
17. Какая функция позволяет вывести все слова в строке с прописных букв? Примените её к строке ‘переходи на тёмную сторону, юзернейм, у нас печеньки!’.
18. Для чего предназначена функция upper?
19. Создайте строку «Заходит лошадь в бар». Примените функции, которые изменяют её выравнивание. Продемонстрируйте результат преподавателю.
20. Для чего используется функция replace()? Приведите пример её использования.

## Лабораторная работа 3. Сложные структуры данных: списки, кортежи

***Цель работы:***

Изучить основные функции для работы со списками и кортежами в языке Python, особенности и области применения этих конструкций.

***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']]
```

**Примечание:**

> *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?

## 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.

***Теоретическая часть.***

**Словари**

Словарь очень похож на список, но порядок элементов в нем не имеет значения, и они выбираются не с помощью смещения. Вместо этого для каждого значения вы указываете связанный с ним уникальный ключ. Таким ключом в основном служит строка, но он может быть объектом одного из неизменяемых типов: булевой переменной, целым числом, числом с плавающей точкой, кортежем, строкой и другими объектами, которые вы увидите далее. Словари можно изменять, что означает, что вы можете добавить, удалить и изменить их элементы, которые имеют вид «ключ — значение».

Если вы работали с другими языками программирования, которые поддерживают только массивы и списки, то полюбите словари.

*Примечание:*

*В других языках программирования словари могут называться ассоциативными массивами, хешами или хеш-таблицей. В Python словарь также называется dict для экономии места.*

**Множества**

Множество похоже на словарь, значения которого опущены. Он имеет только ключи. Как и в случае со словарем, ключи должны быть уникальны. Если вам нужно прикрепить к ключу некую информацию, воспользуйтесь словарем.

Раньше кое-где теорию множеств преподавали в начальной школе наряду с основами математики. На рис. 4.1 вы можете увидеть объединения и пересечения множеств.

Предположим, вы хотите объединить два множества, которые содержат несколько общих ключей. Поскольку множество должно содержать только уникальные значения, объединение двух множеств будет содержать лишь одно включение каждого ключа. Пустое множество — это множество, содержащее ноль элементов.

Основываясь на рис. 4.1, можно сказать, что пустым будет множество, которое содержит женские имена, начинающиеся с буквы «X».

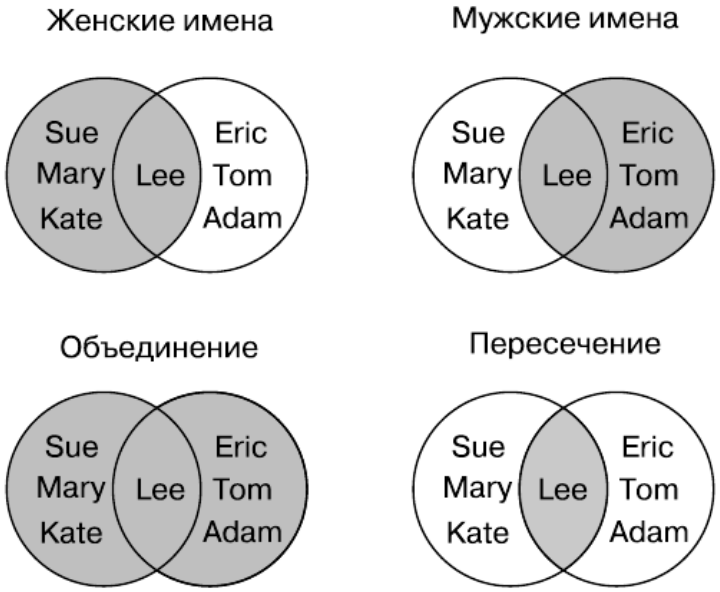



Рис. 4.1 – Базовые операции с множествами

**Оборудование и материалы.**

Персональный компьютер, среда разработки Python.

**Указания по технике безопасности:**

Соответствуют технике безопасности по работе с компьютерной техникой.

## Задания

***Создание словаря с помощью {}***

Чтобы создать словарь, вам нужно обернуть в фигурные скобки ({}) разделенные запятыми пары ключ : значение. Самым простым словарем является пустой словарь, не содержащий ни ключей, ни значений:

```
>>> empty_dict = {}
>>> empty_dict
{}
```

Создадим небольшой юмористический словарь:

```
>>> humorous_dict = {
        "Мыло-Варение" : "Особый джем из мыла",
        "Авто-Ручка" : "Ручка автомобиля",
        "МышеЛовка" : "Очень ловкая мышь",
        }
```

Ввод имени словаря в интерактивный интерпретатор выведет все его ключи и значения:

```
>>> humorous_dict
{'Мыло-Варение': 'Особый джем из мыла', 'Авто-Ручка': 'Ручка автомобиля',
 'МышеЛовка': 'Очень ловкая мышь'}
```

***Примечание:***

> *В Python допускается наличие запятой после последнего элемента списка, кортежа или словаря. Вам также не обязательно использовать выделение пробелами, как сделано в предыдущем примере, когда вы вводите ключи и значения внутри фигурных скобок. Это лишь улучшает читабельность.*

**Преобразование с помощью функции dict()**

Вы можете использовать функцию dict(), чтобы преобразовывать последовательности из двух значений в словари. (Вы иногда можете столкнуться с последовательностями «ключ — значение» вида «Стронций, 90, углерод, 14» или «Вавилон, 5, Глубокий космос, 9».) Первый элемент каждой последовательности применяется как ключ, а второй — как значение.

Для начала рассмотрим небольшой пример, использующий lol (список, содержащий списки, которые состоят из двух элементов):

```
>>> lol = [['a', 'b'], ['c', 'd'],['e', 'f']]
>>> dict(lol)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

> *Помните, что порядок ключей в словаре может быть произвольным, он зависит от того, как вы добавляете элементы.*

Мы могли бы использовать любую последовательность, содержащую последовательности, которые состоят из двух элементов. Рассмотрим остальные примеры.

Список, содержащий двухэлементные кортежи:

```
>>> lot = [('a', 'b'), ('c', 'd'),('e', 'f')]
>>> dict(lot)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Кортеж, включающий двухэлементные списки:

```
>>> tol = (['a', 'b'], ['c', 'd'],['e', 'f'])
>>> dict(tol)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Список, содержащий двухсимвольные строки:

```
>>> los = ['ab', 'cd', 'ef']
>>> dict(los)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

Кортеж, содержащий двухсимвольные строки:

```
>>> tos = ('ab', 'cd', 'ef')
>>> dict(tos)
{'a': 'b', 'c': 'd', 'e': 'f'}
```

***Добавление или изменение элемента с помощью конструкции [ключ]***

Добавить элемент в словарь довольно легко. Нужно просто обратиться к элементу по его ключу и присвоить ему значение. Если ключ уже существует в словаре, имеющееся значение будет заменено новым. Если ключ новый, он и указанное значение будут добавлены в словарь. Здесь, в отличие от списков, вам не нужно волноваться о том, что Python сгенерирует исключение во время присваивания нового элемента, если вы укажете, что этот индекс находится вне существующего диапазона.

Создадим словарь, содержащий список известных студентов факультета Гриффиндор, используя их фамилии в качестве ключей, а имена — в качестве значений:

```
>>> gryffindor = {
        'Поттер' : 'Гарри',
        'Дамблдор' : 'Альбус',
        'Грейнджер' : 'Гермиона',
        'Люпин' : 'Римус',
        'Макгонагал' : 'Минерва',
        'Уизли' : 'Рон',
        }
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона',
'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Рон'}
```

Здесь не хватает Сириуса Блэка. Перед вами попытка анонимного программиста добавить его, однако он ошибся, когда вводил имя:

```
>>> gryffindor ['Блэк'] = 'Юпитер'
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона',
'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Рон', 'Блэк':
'Юпитер'}
```

А вот код другого программиста, который исправил эту ошибку:

```
>>> gryffindor ['Блэк'] = 'Сириус'
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона',
'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Рон', 'Блэк':
'Сириус'}
```

Используя один и тот же ключ ('Блэк'), мы заменили исходное значение 'Юпитер' на 'Сириус'.

Помните, что ключи в словаре должны быть уникальными. Если мы после Уизли Рона внесём Уизли Джинни, то победит последнее значение:

```
>>> gryffindor ['Уизли'] = 'Джинни'
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус'}
```

Сначала мы присвоили значение 'Рон' ключу 'Уизли', а затем заменили его на 'Джинни'.

**Объединение словарей с помощью функции update()**

Вы можете использовать функцию update(), чтобы скопировать ключи и значения из одного словаря в другой.

У нас есть словарь, в котором содержится перечень студентов Гриффиндора.

Кроме того, у нас есть другой словарь, содержащий ещё несколько имён, и называющийся others:

```
>>> others = {'Гриффиндор' : 'Годрик', 'Хагрид' : 'Рубеус'}
```

Теперь появляется еще один анонимный программист, который считает, что члены словаря others должны быть членами gryffindor:

```
>>> gryffindor.update(others)
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус', 'Гриффиндор': 'Годрик', 'Хагрид': 'Рубеус'}
```

Что произойдет, если во втором словаре будут находиться такие же ключи, что и в первом? Победит значение из второго словаря:

```
>>> first = {'a': 1, 'b': 2}
>>> second = {'b': 'platypus'}
>>> first.update(second)
>>> first
{'b': 'platypus', 'a': 1}
```

***Удаление элементов по их ключу с помощью del***

Код нашего анонимного программиста был корректным — технически. Но он не должен был его писать! Члены словаря others, несмотря на свою известность, не закончили факультет. Отменим добавление последних двух элементов:

```
>>> del gryffindor['Гриффиндор']
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус', 'Хагрид': 'Рубеус'}
>>> del gryffindor['Хагрид']
>>> gryffindor
{'Поттер': 'Гарри', 'Дамблдор': 'Альбус', 'Грейнджер': 'Гермиона
', 'Люпин': 'Римус', 'Макгонагал': 'Минерва', 'Уизли': 'Джинни',
'Блэк': 'Сириус'}
```

***Удаление всех элементов с помощью функции clear()***

Чтобы удалить все ключи и значения из словаря, вам следует использовать функцию clear() или просто присвоить пустой словарь заданному имени:

```
>>> gryffindor.clear()
>>> gryffindor
{}
>>> gryffindor = {}
>>> gryffindor
{}
```

***Проверяем на наличие ключа с помощью in***

Если вы хотите узнать, содержится ли в словаре какой-то ключ, используйте ключевое слово in. Снова определим словарь pythons, но на этот раз опустим одного-двух участников. Затем проверим, кого мы добавили:

```
>>> gryffindor = {'Поттер': 'Гарри', 'Дамблдор': 'Альбус',
'Грейнджер': 'Гермиона'}
>>> 'Грейнджер' in gryffindor
True
>>> 'Поттер' in gryffindor
True
```

Вспомнили ли мы о Роне Уизли на этот раз?

```
>>> 'Уизли' in gryffindor
False
```

***Получение элемента словаря  с помощью конструкции [ключ]***

Этот вариант использования словаря - самый распространенный. Вы указываете словарь и ключ, чтобы получить соответствующее значение:

```
>>> gryffindor['Грейнджер']
'Гермиона'
```

Если ключа в словаре нет, будет сгенерировано исключение:

```
>>> gryffindor['Блэк']
Traceback (most recent call last):
  File "<pyshell#271>", line 1, in <module>
    gryffindor['Блэк']
KeyError: 'Блэк'
```

Есть два хороших способа избежать возникновения этого исключения. Первый из них — проверить, имеется ли заданный ключ, с помощью ключевого слова in, что вы уже видели в предыдущем разделе:

```
>>> 'Блэк' in gryffindor
False
```

Второй способ — использовать специальную функцию словаря get(). Вы указываете словарь, ключ и опциональное значение. Если ключ существует, вы получите связанное с ним значение:

```
>>> gryffindor.get('Поттер')
'Гарри'
```

Если такого ключа нет, получите опциональное значение, если указывали его:

```
>>> gryffindor.get('Малфой', 'Не учился на факультете')
'Не учился на факультете'
```

В противном случае будет возвращен объект None (интерактивный интерпретатор не выведет ничего):

```
>>> gryffindor.get('Малфой')
```

***Получение всех ключей с помощью функции keys()***

Вы можете использовать функцию keys(), чтобы получить все ключи словаря. Для следующих примеров мы берем другой словарь:

```
>>> signals = {'Зеленый' : 'Вперёд!', 'Желтый' : 'Беги быстрее',
'Красный' : 'Стоп!'}
>>> signals.keys()
dict_keys(['Зеленый', 'Желтый', 'Красный'])
```

> *Примечание:*
>
> *В Python 2 функция keys() возвращает простой список. В Python 3 эта функция возвращает dict_keys() — итерабельное представление ключей. Это удобно для крупных словарей, поскольку не требует времени и памяти для создания и сохранения списка, которым вы, возможно, даже не воспользуетесь. Но зачастую вам нужен именно список. В Python 3 надо вызвать функцию list(), чтобы преобразовать dict_keys в список:*
>
> *>>>list(signals.keys())*
>
> *['Зеленый', 'Желтый, 'Красный']*
>
> *В Python 3 вам также понадобится использовать функцию list(), чтобы преобразовать результат работы функций values() и items() в обычные списки. Я пользуюсь этой функцией в своих примерах.*

**Получение всех значений с помощью функции values()**

Чтобы получить все значения словаря, используйте функцию values():

```
>>> list(signals.values())
['Вперёд!', 'Беги быстрее', 'Стоп!']
```

**Получение всех пар «ключ — значение» с помощью функции items()**

Когда вам нужно получить все пары «ключ — значение» из словаря, используйте функцию items():

```
>>> list(signals.items())
[('Зеленый', 'Вперёд!'), ('Желтый', 'Беги быстрее'), ('Красный',
'Стоп!')]
```

Каждая пара будет возвращена как кортеж вроде ('Зеленый', 'Вперёд!').

**Присваиваем значения с помощью оператора =, копируем их с помощью функции copy()**

Как и в случае со списками, если вам нужно внести в словарь изменение, оно отразится для всех имен, которые ссылаются на него.

```
>>> signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Сто
п!'}
>>> save_signals = signals
>>> signals['Голубой'] = 'Конфуз'
>>> save_signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Сто
п!', 'Голубой': 'Конфуз'}
```

Чтобы скопировать ключи и значения из одного словаря в другой и избежать этого, вы можете воспользоваться функцией copy():

```
>>> signals = {'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Крас
ный': 'Стоп!'}
>>> original_signals = signals.copy()
>>> signals['Голубой'] = 'Неожиданно'
>>> signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Стоп!',
'Голубой': 'Неожиданно'}
>>> original_signals
{'Зеленый': 'Вперёд!', 'Желтый': 'Беги быстрее', 'Красный': 'Стоп!'}
```

## Множества

### Создание множества с помощью функции set()

Чтобы создать множество, вам следует использовать функцию set() или разместить в фигурных скобках одно или несколько разделенных запятыми значений, как показано здесь:

```
>>> empty_set = set()
>>> empty_set
set()
>>> even_numbers = {0,2,4,6,8}
>>> even_numbers
{0, 2, 4, 6, 8}
>>> odd_numbers = {1,3,5,7,9}
>>> odd_numbers
{1, 3, 5, 7, 9}
```

Как и в случае со словарем, порядок ключей в множестве не имеет значения.

**Примечание:**

> Поскольку пустые квадратные скобки [] создают пустой список, вы могли бы рассчитывать на то, что пустые фигурные скобки {} создают пустое множество. Вместо этого пустые фигурные скобки создают пустой словарь. Именно поэтому интерпретатор выводит пустое множество как set() вместо {}. Почему так происходит? Словари появились в Python раньше и успели захватить фигурные скобки в свое распоряжение.

### Преобразование других типов данных с помощью функции set()

Вы можете создать множество из списка, строки, кортежа или словаря, потеряв все повторяющиеся значения.

Для начала взглянем на строку, которая содержит более чем одно включение некоторых букв:

```
>>> set('letters')
{'e', 't', 's', 'l', 'r'}
```

Обратите внимание на то, что множество содержит только одно включение букв «e» или «t», несмотря на то, что в слове letters по два включения каждой из них.

Создадим множество из списка:

```
>>> set(['Железный человек', 'Капитан Америка', 'Соколиный глаз', 'Халк'])
{'Халк', 'Капитан Америка', 'Железный человек', 'Соколиный глаз'}
```

А теперь из кортежа:

```
>>> set(('Флеш', 'Аквамен', 'Бетмен', 'Зелёная стрела'))
{'Зелёная стрела', 'Аквамен', 'Флеш', 'Бетмен'}
```

Когда вы передаете функции set() словарь, она возвращает только ключи:

```
>>> set({'Яблоко' : 'Зелёное', 'Апельсин' : 'Оранжевый', 'Грейпфрут' :
'Оранжевый'})
{'Яблоко', 'Грейпфрут', 'Апельсин'}
```

**Проверяем на наличие значения с помощью ключевого слова in**

Такое использование множеств самое распространенное. Мы создадим словарь, который называется recipes. Каждый ключ будет названием коктейля, а соответствующие значения — множествами ингредиентов:

```
recipes = {
    'Борщ' : {'Капуста', 'Картошка', 'Помидоры'},
    'Салат Весенний' : {'Помидоры', 'Огурцы', 'Сметана'},
    'Луковый суп' : {'Бульон', 'Лук'},
    'Омлет' : {'Яйца', 'Помидоры'},
    'Пельмени' : {'Фарш', 'Мука'},
    'Смузи' : {'Банан', 'Клубника', 'Йогурт'}
    }
```

Несмотря на то что и словарь, и множества окружены фигурными скобками ({ и }), множество — это всего лишь последовательность значений, а словарь — это набор пар «ключ — значение».

Какой из рецептов содержит помидоры? (Обратите внимание на то, что для выполнения этих проверок я заранее демонстрирую использование ключевых слов for, if, and и or, которые будут рассмотрены только в следующей работе.)

```
>>> for name, components in recipes.items():
        if 'Помидоры' in components:
                print(name)


Борщ
Салат Весенний
Омлет
```

Мы хотим съесть блюдо с помидорами, но не переносим лактозу, а на клубнику у нас аллергия:

```
>>> for name, components in recipes.items():
        if 'Помидоры' in components and not ('Сметана' in components
or 'Клубника' in components):
                print(name)


Борщ
Омлет
```

Перепишем этот пример чуть более сжато в следующем разделе.

**Комбинации и операторы**

Что, если вам нужно проверить наличие сразу нескольких значений множества?

Предположим, вы хотите найти любой рецепт, содержащий много крахмала (как в картофеле или бананах). Для этого мы используем оператор пересечения множеств (&):

```
>>> for name, components in recipes.items():
        if components & {'Картошка', 'Банан'}:
            print(name)


Борщ
Смузи
```

Результатом работы оператора & является множество, содержащее все элементы, которые находятся в обоих сравниваемых списках. Если ни один из заданных ингредиентов не содержится в предлагаемых рецептах, оператор & вернет пустое множество. Этот результат можно считать равным False.

Теперь перепишем пример из предыдущего раздела, в котором мы хотели помидоров, не смешанных со сметаной или клубникой:

```
>>> for name, components in recipes.items():
    if 'Помидоры' in components and not components &{'Сметана', 'Клубника'}:
        print(name)


Борщ
Омлет
```

Сохраним множества ингредиентов для двух рецептов в переменных, чтобы нам не пришлось набирать много текста в дальнейших примерах:

```
>>> Salad = recipes['Салат Весенний']
>>> Borsh = recipes['Борщ']
```

В следующих примерах демонстрируется использование операторов множеств. В одних из них демонстрируется применение особой пунктуации, в других — особых функций, в третьих — и того и другого. Мы будем использовать тестовые множества a (содержит элементы 1 и 2) и b (содержит элементы 2 и 3)

```
>>> a = {1, 2}
>>> b = {2, 3}
```

Пересечение множеств (члены обоих множеств) можно получить с помощью особого пунктуационного символа & или функции множества intersection(), как показано здесь

```
>>> a & b
{2}
>>> a.intersection(b)
{2}
```

В этом фрагменте используются сохраненные нами переменные:

```
>>> Salad & Borsh
{'Помидоры'}
```

В этом примере мы получаем объединение (члены обоих множеств), используя оператор | или функцию множества union():

```
>>> a | b
{1, 2, 3}
>>> a.union(b)
{1, 2, 3}
```

Обеденная версия:

```
>>> Salad | Borsh
{'Сметана', 'Огурцы', 'Капуста', 'Картошка', 'Помидоры'}
```

Разность множеств (члены только первого множества, но не второго) можно получить с помощью символа – или функции difference():

```
>>> a - b
{1}
>>> a.difference(b)
{1}
>>> Borsh - Salad
{'Капуста', 'Картошка'}
```

Самыми распространенными операциями с множествами являются объединение, пересечение и разность. Для полноты картины я включил в этот раздел и остальные операции, но вы, возможно, никогда не будете их использовать.

Для выполнения исключающего ИЛИ (элементы или первого, или второго множества, но не общие) используйте оператор ^ или функцию symmetric_difference():

```
>>> a ^ b
{1, 3}
>>> a.symmetric_difference(b)
{1, 3}
>>> Borsh ^ Salad
{'Сметана', 'Капуста', 'Огурцы', 'Картошка'}
```

В последнем примере показано, в чём разница между борщём и салатом ☺.

Вы можете проверить, является ли одно множество подмножеством другого (все члены первого множества являются членами второго), с помощью оператора <= или функции issubset():

```
>>> a = a.union(b)
>>> a
{1, 2, 3}
>>> a <= b
False
>>> b <= a
True
```

```
>>> a <= b
False
>>> a.issubset(b)
False
```

Является ли любое множество подмножеством самого себя? Ага.

```
>>> a <= a
True
>>> a.issubset(a)
True
```

Для того чтобы одно подмножество стало подмножеством второго, второе множество должно содержать все члены первого и несколько других. Определяется это с помощью оператора <:

```
>>> a < b
False
>>> a < a
False
>>> b < a
True
>>> Borsh < Salad
False
```

Множество множествпротивоположно подмножеству (все члены второго множества являются также членами первого). Для определения этого используется оператор >= или функция issuperset():

```
>>> a >= b
True
>>> a.issuperset(b)
True
```

Любое множество является множеством множеств самого себя:

```
>>> a >= a
True
>>> a.issuperset(a)
True
```

И наконец, вы можете найти собственное множество множеств (первое множество содержит все члены второго и несколько других) с помощью оператора >:

```
>>> b > a
False
>>> a > b
True
>>> Borsh > Salad
False
```

Множество не может быть собственным множеством множеств самого себя:

```
>>> a > a
False
```

**Сравнение структур данных**

Напомню, список создается с помощью квадратных скобок ([]), кортеж — с помощью запятых, а словарь — с помощью фигурных скобок ({}). Во всех случаях вы получаете доступ к отдельному элементу с помощью квадратных скобок:

```
>>> frend_list = ['Чендлер', 'Росс', 'Джо']
>>> frend_tuple = 'Чендлер', 'Росс', 'Джо'
>>> frend_dict = {'Бинк' : 'Чендлер', 'Геллер' : 'Росс', 'Трибиани' : 'Джо'}
>>> frend_list[2]
'Джо'
>>> frend_tuple[2]
'Джо'
>>> frend_dict['Трибиани']
'Джо'
```

Для списка и кортежа значение, находящееся в квадратных скобках, является целочисленным смещением. Для словаря же оно является ключом. Для всех троих результатом будет значение.

***Создание крупных структур данных***

Ранее мы работали с простыми булевыми значениями, числами и строками. Теперь же мы работаем со списками, кортежами, множествами и словарями. Вы можете объединить эти встроенные структуры данных в собственные структуры, более крупные и сложные. Начнем с трех разных списков:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> pythons = ['Chapman', 'Cleese', 'Gilliam', 'Jones', 'Palin']
>>> stooges = ['Moe', 'Curly', 'Larry']
```

Мы можем создать кортеж, который содержит в качестве элементов каждый из этих списков:

```
>>> tuple_of_lists = friends, physicists, gryffindor
>>> tuple_of_lists
(['Чендлер', 'Росс', 'Джо'], ['Леонард', 'Шелдон', '
Радж', 'Говард'], ['Гарри', 'Рон', 'Гермиона'])
```

Можем также создать список, который содержит три списка:

```
>>> list_of_lists = [friends, physicists, gryffindor]
>>> list_of_lists
[['Чендлер', 'Росс', 'Джо'], ['Леонард', 'Шелдон', 'Радж',
'Говард'], ['Гарри', 'Рон', 'Гермиона']]
```

Наконец, создадим словарь из списков. В этом примере используем название группы комиков в качестве ключа, а список ее членов — в качестве значения:

```
>>> dict_of_lists = {'Друзья' : friends, 'Физики' :
physicists, 'Гриффиндорцы' : gryffindor}
>>> dict_of_lists
{'Друзья': ['Чендлер', 'Росс', 'Джо'], 'Физики': ['Л
еонард', 'Шелдон', 'Радж', 'Говард'], 'Гриффиндорцы'
: ['Гарри', 'Рон', 'Гермиона']}
```

Вас ограничивают только сами типы данных. Например, ключи словаря должны быть неизменяемыми, поэтому список, словарь или множество не могут быть ключом для другого словаря. Но кортеж может быть ключом. Например, вы можете создать алфавитный указатель достопримечательностей, основываясь на GPS-координатах

(широте, долготе и высоте):

```
>>> houses = {
        (44.79, -93.14, 285): 'My House',
        (38.89, -77.03, 13): 'The White House'
        }
```

## Упражнения

Вы познакомились с более сложными структурами данных: списками, кортежами, словарями и множествами. Используя их и типы данных, описанные в предыдущих работах (числа и строки), вы можете представить множество элементов реального мира.

1. Создайте список years_list, содержащий год, в который вы родились, и каждый последующий год вплоть до вашего пятого дня рождения. Например, если вы родились в 1980 году, список будет выглядеть так: years_list = [1980, 1981, 1982, 1983, 1984, 1985].
2. В какой из годов, содержащихся в списке years_list, был ваш третий день рождения? Помните, в первый год вам было 0 лет.
3. В какой из годов, перечисленных в списке years_list, вам было больше всего лет?
4. Создайте список things, содержащий три элемента: "mozzarella", "cinderella", "salmonella".
5. Напишите с большой буквы тот элемент списка things, который относится к человеку, а затем выведите список. Изменился ли элемент списка?
6. Переведите сырный элемент списка things в верхний регистр целиком и выведите список.
7. Удалите болезнь из списка things, получите Нобелевскую премию и затем выведите список на экран.
8. Создайте список, который называется surprise и содержит элементы 'Groucho', 'Chico' и 'Harpo'.
9. Напишите последний элемент списка surprise со строчной буквы, затем обратите его и напишите с прописной буквы.
10. Создайте англо-французский словарь, который называется e2f, и выведите его на экран. Вот ваши первые слова: dog/chien, cat/chat и walrus/morse.
11. Используя словарь e2f, выведите французский вариант слова walrus.
12. Создайте французско-английский словарь f2e на основе словаря e2f. Используйте метод items.
13. Используя словарь f2e, выведите английский вариант слова chien.
14. Создайте и выведите на экран множество английских слов из ключей словаря e2f.
15. Создайте многоуровневый словарь life. Используйте следующие строки для ключей верхнего уровня: 'животные', 'растения' и 'другое'. Сделайте так, чтобы ключ 'животные' ссылался на другой словарь, имеющий ключи 'котики', 'осьминоги' и 'эму'. Сделайте так, чтобы ключ 'котики' ссылался на список строк со значениями 'Оппенгеймер', 'Эйнштейн' и 'Лапусик'. Остальные ключи должны ссылаться на пустые словари.
16. Выведите на экран высокоуровневые ключи словаря life.
17. Выведите на экран ключи life['животные'].
18. Выведите значения life['животные']['котики'].

### ***Контрольные вопросы***

1. В чем отличие словарей от других структур данных в Python?
2. Как создать словарь?
3. Как создать словарь из кортежа? Из списка? Из списка двухсимвольных строк?

4. Как добавить данные в словарь?
5. Какие требования предъявляются значению ключа в словаре?
6. Что произойдет, если новый ключ совпадет с уже имеющимся?
7. Как объединить два словаря?
8. Как удалить элементы из словаря?
9. Как очистить словарь целиком?
10. Как получить элемент словаря по ключу?
11. Какая функция ищет в словаре заданный элемент и возвращает заданное значение, если элемент не найден?
12. Как получить ключи словаря?
13. Как получить значения словаря?
14. Как скопировать значения из одного словаря в другой и избежать их связывания?
15. В чём особенность множеств?
16. Как создать пустое множество?
17. Что такое пересечение множеств? Как его найти? Какой будет получен тип данных?
18. Что такое объединение множеств? Как его найти?
19. Что такое разность множеств? Как ее получить?
20. Что такое «исключающее ИЛИ»? Как его получить?
21. Как проверить, является ли одно множество подмножеством другого?
22. Как создать кортеж из списков?
23. Как создать список списков?
24. Как создать словарь из списков?
25. Какие структуры можно использовать в качестве ключа словаря?

## Лабораторная работа 5. Структура кода в языке Python. Линейные алгоритмы и алгоритмы ветвления.

***Цель работы:***

Изучить способы оформления структуры кода в языке Python. Научиться применять линейные алгоритмы и алгоритмы ветвления и выбора.

***Теоретическая часть.***

В предыдущих работах вы увидели множество примеров данных, но практически не работали с ними. В большинстве примеров использовался интерактивный интерпретатор, а сами они были довольно короткими. Теперь вы увидите, как структурировать код Python, а не только данные.

В большинстве языков программирования символы вроде фигурных скобок ({ и }) или ключевые слова вроде begin и end применяется для того, чтобы разбить код на разделы. В этих языках хорошим тоном является использование отбивки пробелами, чтобы сделать программу более удобочитаемой для себя и других. Существуют даже инструменты, которые помогут красиво выстроить ваш код.

Гвидо ван Россум при разработке Python решил, что выделения пробелами будет достаточно, чтобы задать структуру программы и избежать ввода всех этих скобок. Python отличается от других языков тем, что пробелы в нем используются для того, чтобы задать структуру программы. Этот аспект новички замечают одним из первых, и он может показаться странным для тех, кто уже работал с другими языками программирования. Однако по прошествии некоторого времени это начинает казаться естественным и вы перестаете это замечать. Вы даже привыкнете к тому, что делаете больше, набирая

меньше текста.

Комментируем с помощью символа #

Комментарий — это фрагмент текста в вашей программе, который будет проигнорирован интерпретатором Python. Вы можете использовать комментарии, чтобы дать пояснение близлежащего кода, сделать какие-то пометки для себя, или для чего-то еще. Комментарий помечается символом #; все, что находится после # до конца текущей строки, является комментарием. Обычно комментарий располагается на отдельной строке, как показано здесь:

```
>>> # 60 с/мин * 60 мин/ч * 24 ч/день
>>> seconds_per_day = 86400
```

Или на той же строке, что и код, который нужно пояснить:

```
>>> seconds_per_day = 86400 # 60 с/мин * 60 мин/ч * 24 ч/день
```

Символ # имеет много имен: хеш, шарп, фунт или устрашающее октоторп. Как бы вы его ни назвали, его эффект действует только до конца строки, на кото рой он располагается.

Python не дает возможности написать многострочный комментарий. Вам нужно явно начинать каждую строку или раздел комментария с символа #:

```
>>> # Я могу сказать здесь всё, даже если Python это не нравится,
>>> # поскольку я защищен крутым
>>> # октоторпом.
```

Однако если октоторп находится внутри текстовой строки, он становится простым символом #:

```
>>> print("Без комментариев: кавычки делают октоторп # безвредным")
Без комментариев: кавычки делают октоторп # безвредным
```

**Оборудование и материалы.**

Персональный компьютер, среда разработки Python.

**Указания по технике безопасности:**

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

***Продлеваем строки с помощью символа ***

Любая программа становится более удобочитаемой, если ее строки относительно короткие. Рекомендуемая (но не обязательная) максимальная длина строки равна 80 символам. Если вы не можете выразить свою мысль в рамках 80 символов, воспользуйтесь символом возобновления \. Просто поместите его в конце строки, и дальше Python будет действовать так, будто это все та же строка.

Например, если бы я хотел создать длинную строку из нескольких коротких, я мог бы сделать это пошагово:

```
>>> alphabet = ''
>>> alphabet += 'abcdefg'
>>> alphabet += 'hijklmnop'
>>> alphabet += 'qrstuv'
>>> alphabet += 'wxyz'
>>> alphabet
'abcdefghijklmnopqrstuvwxyz'
```

Или же за одно действие, использовав символ *continuation*:

```
>>> alphabet = ''
>>> alphabet = 'abcdefg' + \
            'hijklmnop' + \
            'qrstuv' + \
            'wxyz'
```

Продлить строку может быть необходимо, если выражение располагается на

нескольких строках:

```
>>> 1 + 2 + \
  3
6
>>>
```

***Сравниваем выражения с помощью операторов if, elif и else***

До этого момента мы говорили только о структурах данных. Теперь же наконец готовы сделать первый шаг к рассмотрению структур кода, которые вводят данные в программы. (Вы уже могли получить представление о них в лабораторной работе 4, в разделе о множествах.) В качестве первого примера рассмотрим небольшую программу, которая проверяет значение булевой переменной disaster и выводит подходящий комментарий:

```
>>> disaster = True
>>> if disaster:
        print("Караул!")
else:
        print("Бугагашенька!")


Караул!
>>>
```

Строки if и else в Python являются операторами, которые проверяют, является ли значение выражения (в данном случае переменной disaster) равным True. Помните, print() — это встроенная в Python функция для вывода информации, как правило, на ваш экран.

> ***Примечание:***
>
> *Если вы работали с другими языками программирования, обратите внимание на то, что при проверке if вам не нужно ставить скобки. Не нужно писать что-то вроде if (disaster == True).*
>
> *В конце строки следует поставить двоеточие (:). Если вы, как и я, иногда забываете ставить двоеточие, Python выведет сообщение об ошибке.*

Каждая строка print() отделена пробелами под соответствующей проверкой. Здесь использовано четыре пробела для того, чтобы выделить каждый подраздел. Хотя вы можете использовать любое количество пробелов, Python ожидает, что внутри одного раздела будет применяться одинаковое количество пробелов. Рекомендованный стиль — PEP-8 (http://bit.ly/pep-8) — предписывает использовать четыре пробела. Не применяйте табуляцию или сочетание табуляций и пробелов — это мешает подсчитывать отступы.

Все выполненные в этом примере действия выполняют следующее:

1. Присвоили булево значение True переменной disaster.
2. Произвели условное сравнение с помощью операторов if и else, выполняя разные фрагменты кода в зависимости от значений переменной disaster.
3. Вызвали функцию print(), чтобы вывести текст на экран.

Можно организовывать проверку в проверке столько раз, сколько вам нужно:

```
>>> furry = True
>>> small = True
>>> if furry:
        if small:
                print("Это котэ")
        else:
                print("Это медведь")
else:
        if small:
                print("Это рыба")
        else:
                print("Это человек. Или лысый медведь.")


Это котэ
>>>
```

В Python отступы определяют, какие разделы if и else объединены в пару. Наша первая проверка обращалась к переменной furry. Поскольку ее значение равно True, Python переходит к выделенной таким же количеством пробелов проверке if small.

Поскольку мы указали значение переменной small равным True, проверка вернет результат True. Это заставит Python вывести на экран строку *Это котэ*.

Если необходимо проверить более двух вариантов, используйте операторы if, elif (это значит else if — «иначе если») и else:

```
>>> color = "Терракотовый"
>>> if color == "Красный":
        print("Это помидор")
elif color =="Зелёный":
        print("Это зелёный перец")
elif color == "Нежно-пурпурный":
        print("Таких овощей я не знаю")
else:
        print("Я никогда не слышал цвет ", color)


Я никогда не слышал цвет  Терракотовый
```

В предыдущем примере мы проверяли равенство с помощью оператора ==. В Python используются следующие операторы сравнения:

- равенство (==);
- неравенство (!=);
- меньше (<);
- меньше или равно (<=);
- больше (>);
- больше или равно (>=);
- включение (in …).

Эти операторы возвращают булевы значения True или False. Взглянем на то, как они работают, но сначала присвоим значение переменной x:

Выполним несколько проверок:

```
>>> x = 7
>>> x == 5
False
>>> x == 7
True
>>> 5 < x
True
>>> x < 10
True
```

Обратите внимание на то, что для проверки на равенство используются два знака «равно» (==); помните, что один знак «равно» применяется для присваивания значения переменной.

Если вам нужно выполнить несколько сравнений одновременно, можете использовать булевы операторы and, or и not, чтобы определить итоговый двоичный результат.

Булевы операторы имеют более низкий приоритет, нежели фрагменты кода, которые они сравнивают. Это значит, что сначала высчитывается результат фрагментов, а затем они сравниваются. В данном примере из-за того, что мы устанавливаем значение x равным 7, проверка 5 < x возвращает значение True и проверка x < 10 также возвращает True, поэтому наше выражение преобразуется в True and True:

```
>>> 5 < x and x < 10
True
```

Однако, самый простой способ избежать путаницы — использовать круглые скобки:

```
>>> (5 < x) and (x < 10)
True
```

Рассмотрим некоторые другие проверки:

```
>>> 5 < x or x < 10
True
>>> 5 < x and x > 10
False
>>> 5 < x and not x > 10
True
```

Если вы используете оператор and для того, чтобы объединить несколько проверок, Python позволит вам сделать следующее:

```
>>> 5 < x < 10
True
```

Это выражение аналогично проверкам 5 < x и x < 10. Вы также можете писать более длинные сравнения:

```
>>> 5 < x < 10 < 999
True
```

Что есть истина? Что, если элемент, который мы проверяем, не является булевым? Чем Python считает True и False?

Значение false не обязательно явно означает False. Например, к False приравниваются все следующие значения:

- булева переменная False;
- значение None;
- целое число 0;
- число с плавающей точкой 0.0;
- пустая строка (' ');
- пустой список ([]);
- пустой кортеж (());
- пустой словарь ({});
- пустое множество (set()).

Все остальные значения приравниваются к True. Программы, написанные на

Python, используют это определение истинности (или, как в данном случае, ложности), чтобы выполнять проверку на пустоту структуры данных наряду с проверкой на равенство непосредственно значению False:

```
>>> some_list = []
>>> if some_list:
        print("Тут что-то есть!")
else:
        print("Тут пусто!")


Тут пусто!
```

Если вы выполняете проверку для выражения, а не для простой переменной, Python оценит его значение и вернет булев результат. Поэтому, если вы введете следующее:

```
>>> if color == "Красный":
```

Python оценит выражение color == "red". В нашем примере мы присвоили переменной color значение "Терракотовый", поэтому значение выражения color == "red" равно False и Python перейдет к следующей проверке:

```
elif color =="Зелёный":
```

***Контрольные вопросы***

1. Как можно продлить строку? Связать несколько строк в одну?
2. Какие операторы сравнения в Python вы знаете?
3. Какова структура оператора if?
4. Как записать в условии знак равенства?
5. Как записать в условии знак неравенства?
6. Как задать условие «меньше либо равно», «больше либо равно»?
7. Как в условии можно проверить включение?
8. Что возвращает оператор сравнения?
9. Как и для чего используются булевы операторы and, or, not?
10. Какой приоритет имеют булевы операторы по сравнению с операторами сравнения?
11. Что произойдет, если элемент, который проверяется в условии, не будет булевого типа?

## Лабораторная работа 6. Работа с циклами в Python

***Цель работы:***

Изучить команды создания циклов в языке Python. Изучить различия между циклами while и for и области их применения. Изучить применение команд break и continue.

***Теоретическая часть.***

Все языки программирования содержат какую-нибудь конструкцию цикла. В большей части языков есть больше одной такой конструкции. В языке Python есть два типа циклов:

- цикл for
- цикл while

Циклы используются в тех случаях, когда нам нужно сделать что-нибудь много раз.

Нередко вам придется выполнить какую-нибудь операцию (или ряд операций) в части данных снова и снова. Тут то и вступают в силу циклы. Благодаря им становится возможно максимально упростить данный вопрос. Давайте подробно разберём, как работают эти структуры!

***Цикл while***

While - один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие цикла истинно.

```
>>> i=5
>>> while i<15:
        print(i)
        i+=2


5
7
9
11
13
```

***Цикл for***

Цикл for уже немного сложнее, чуть менее универсальный, но выполняется гораздо быстрее цикла while. Этот цикл проходится по любому итерируемому объекту (например строке или списку), и во время каждого прохода выполняет тело цикла.

```
>>> for i in "hello world":
        print(i*2, end='')


hheellllloo  wwoorrlldd
```

***Операторcontinue***

Операторcontinueначинаетследующийпроходцикла, минуяоставшеесятелоцикла (forилиwhile)

```
>>> for i in 'hello world':
        if i=='o':
                continue
        print(i*2, end='')


hheellll  wwrrlldd
```

***Операторbreak***

Операторbreakдосрочнопрерываетцикл.

```
>>> for i in 'hello world':
        if i=='o':
                break
        print(i*2, end='')


hheellll
```

***Команда else***

Слово else, примененное в цикле for или while, проверяет, был ли произведен выход из цикла инструкцией break, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи break.

```
>>> for i in 'hello world':
        if i=='a':
                break
else:
                print('Буквы a в строке нет')


Буквы a в строке нет
```

**Оборудование и материалы.**
Персональный компьютер, среда разработки Python.
**Указания по технике безопасности:**
Соответствуют технике безопасности по работе с компьютерной техникой.

***Задания***

***Повторяем действия с помощью while***

Проверки с помощью if, elif и else выполняются последовательно. Иногда нам нужно выполнить какие-то операции более чем один раз. Нам нужен цикл, и простейшим вариантом циклов в Python является while. Попробуйте запустить с помощью интерактивного интерпретатора следующий пример — это простейший цикл, который выводит на экран значения от 1 до 5:

```
>>> while count <= 5:
        print(count)
        count += 1


1
2
3
4
5
```

Сначала мы присваиваем значение 1 переменной count. Цикл while сравнивает значение переменной count с числом 5 и продолжает работу, если значение переменной count меньше или равно 5. Внутри цикла мы выводим значение переменной count, а затем увеличиваем его на 1 с помощью выражения count += 1. Python возвращается к верхушке цикла и снова сравнивает значение переменной count с числом 5. Значение переменной count теперь равно 2, поэтому содержимое цикла while выполняется снова и переменная count увеличивается до 3.

Это продолжается до тех пор, пока переменная count не будет увеличена с 5 до 6 в нижней части цикла. Во время очередного возврата наверх цикла проверка count<= 5 вернет значение False и цикл while закончится. Python перейдет к выполнению следующих строк.

***Прерываем цикл с помощью break***

Если вы хотите, чтобы цикл выполнялся до тех пор, пока что-то не произойдет, но вы не знаете точно, когда это событие случится, можете воспользоваться бесконечным циклом, содержащим оператор break. В этот раз мы считаем строку с клавиатуры с помощью функции input(), а затем выведем ее на экран, сделав первую букву прописной. Мы прервем цикл, когда будет введена строка, содержащая только букву «q»:

```
>>> while True:
        stuff = input("Строка с заглавной буквы [или q для выхода]: ")
        if stuff == "q":
                break
        print(stuff.capitalize())


Строка с заглавной буквы [или q для выхода]: тест
Тест
Строка с заглавной буквы [или q для выхода]: да, всё работает
Да, всё работает
Строка с заглавной буквы [или q для выхода]: q
```

***Пропускаем итерации с помощью continue***

Иногда вам нужно не прерывать весь цикл, а только пропустить по какой-то причине одну итерацию. Рассмотрим воображаемый пример: считаем целое число, выведем на экран его значение в квадрате, если оно нечетное, и пропустим его, если оно четное. И вновь для выхода из цикла используем строку "q":

```
>>> while True:
        value = input("Целое, пожалуйста [или q для выхода]: ")
        if value == 'q':
                break
        number = int(value)
        if number % 2 == 0:
                continue
        print(number, "квадрат = ", number*number)


Целое, пожалуйста [или q для выхода]: 2
Целое, пожалуйста [или q для выхода]: 5
5 квадрат =  25
Целое, пожалуйста [или q для выхода]: 3
3 квадрат =  9
Целое, пожалуйста [или q для выхода]: 80
Целое, пожалуйста [или q для выхода]: q
>>>
```

***Проверяем, завершился ли цикл заранее, с помощью else***

Если цикл while завершился нормально (без вызова break), управление передается в опциональный блок else. Вы можете использовать его в цикле, где выполняете некоторую проверку и прерываете цикл, как только проверка успешно выполняется. Блок else выполнится в том случае, если цикл while будет пройден полностью, но искомый объект не будет найден:

```
>>> numbers = [1, 3, 5]
>>> position = 0
>>> while position < len(numbers):
        number = numbers[position]
        if number % 2 == 0:
                print('Найдено чётное число: ', number)
                break
        position +=1
else: #Выхода не произошло
        print('Чётное число в списке отсутствует')


Чётное число в списке отсутствует
```

### *Выполняем итерации с помощью for*

В Python итераторы часто используются по одной простой причине. Они позволяют вам проходить структуры данных, не зная, насколько эти структуры велики и как реализованы. Вы даже можете пройти по данным, которые были созданы во время работы программы, что позволяет обработать потоки данных, которые в противном случае не поместились бы в память компьютера.

Вполне возможно пройти по последовательности таким образом:

```
>>> hobbits = ['Фродо', 'Сэм', 'Мерри', 'Пин']
>>> current = 0
>>> while current < len(hobbits):
        print(hobbits[current])
        current +=1


Фродо
Сэм
Мерри
Пин
```

Однако существует более характерный для Python способ решения этой задачи:

```
>>> for hobbit in hobbits:
        print(hobbit)


Фродо
Сэм
Мерри
Пин
```

Списки вроде rabbits являются одними из итерабельных объектов в Python наряду со строками, кортежами, словарями и некоторыми другими элементами. Итерирование по кортежу или списку возвращает один элемент за раз. Итерирование по строке возвращает один символ за раз, как показано здесь:

```
>>> word = 'котэ'
>>> for letter in word:
        print(letter)


к
о
т
э
```

Итерирование по словарю (или его функции keys()) возвращает ключи. В этом примере ключи являются номерами пальцев на руках:

```
>>> fingers = {'Первый' : 'Большой', 'Второй' : 'Указательный', 'Третий' :
'Средний', 'Четвёртый' : 'Безымянный', 'Пятый' : 'Мизинец'}
>>> for finger in fingers: # или for finger in fingers.keys():
        print(finger)


Первый
Второй
Третий
Четвёртый
Пятый
>>>
```

Чтобы итерировать по значениям, а не по ключам, следует использовать функцию values():

```
>>> for value in fingers.values():
        print(value)


Большой
Указательный
Средний
Безымянный
Мизинец
```

Чтобы вернуть как ключ, так и значение словаря, вы можете использовать функцию items():

```
>>> for item in fingers.items():
        print(item)


('Первый', 'Большой')
('Второй', 'Указательный')
('Третий', 'Средний')
('Четвёртый', 'Безымянный')
('Пятый', 'Мизинец')
>>>
```

Помните, что можете присвоить значение кортежу за один шаг. Для каждого кортежа, возвращенного функцией items(), присвойте первое значение (ключ) переменной card, а второе (значение) — переменной contents:

```
>>> for number, finger in fingers.items():
        print('Палец ', number, 'называется', finger)


Палец  Первый называется Большой
Палец  Второй называется Указательный
Палец  Третий называется Средний
Палец  Четвёртый называется Безымянный
Палец  Пятый называется Мизинец
```

***Прерываем цикл с помощью break***

Ключевое слово break в цикле for прерывает этот цикл точно так же, как и цикл while.

***Пропускаем итерации с помощью continue***

Добавление ключевого слова continue в цикл for позволяет перейти на следующую

итерацию цикла, как и в случае с циклом while.

***Проверяем, завершился ли цикл заранее, с помощью else***

Как и в цикле while, в for имеется опциональный блок else, который проверяет, выполнился ли цикл for полностью. Если ключевое слово break не было вызвано, будет выполнен блок else.

Это полезно, если вам нужно убедиться в том, что предыдущий цикл выполнился полностью, вместо того чтобы рано прерваться. Цикл for в следующем примере выводит на экран название сыра и прерывается, если сыра в магазине не найдется:

```
>>> cheeses = []
>>> for cheese in cheeses:
        print('Этот магазин имеет отличный ', cheese)
        break
else: # Отсутствие прерывания означает, что сыр не найден
        print('Сыра в этом магазине нет, не так ли?')


Сыра в этом магазине нет, не так ли?
>>>
```

> Примечание:
>
> Как и в цикле while, в цикле for использование блока else может показаться нелогичным. Можно рассматривать цикл for как поиск чего-то, в таком случае else будет вызываться, если вы ничего не нашли. Чтобы получить тот же эффект без блока else, используйте переменную, которая будет показывать, нашелся ли искомый элемент в цикле for, как здесь:
>
> ```
> >>> cheeses = []
> >>> found_one = False
> >>> for cheese in cheeses:
>         found_one = True
>         print('В этом магазине есть отличный сыр ', cheese)
>         break
>
> >>> if not found_one:
>         print('Наверное, это не продуктовый магазин.')
>
>
> Наверное, это не продуктовый магазин.
> >>>
> ```

***<u>Контрольные вопросы</u>***

1. Для чего используются циклы?
2. Опишите особенности применения цикла while.
3. В чём разница между elifи else?
4. Для чего используется оператор break?
5. Для чего используется оператор continue?
6. Как можно проверить, завершился ли цикл досрочно?
7. В чём особенности синтаксиса цикла for?
8. Как выполнить итерирование по словарю? Как вывести ключи?
9. Как выполнить итерирование словаря и вывести его значения?
10. Как вывести и ключи и значения словаря?
11. Как используется прерывание в цикле for? Для чего?

## 4. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

## 5. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ, ОПРЕДЕЛЯЮЩИЕ ПРОЦЕДУРЫ ОЦЕНИВАНИЯ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ И (ИЛИ) ОПЫТА ДЕЯТЕЛЬНОСТИ, ХАРАКТЕРИЗУЮЩИХ ЭТАПЫ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ

Текущая аттестация студентов проводится преподавателями, ведущими лабораторные занятия по дисциплине, в следующих формах: отчет письменный по заданию преподавателя, контрольная работа.

Допуск к лабораторным работам происходит при наличии у студентов печатного варианта отчета. Защита отчета проходит в форме доклада студента по выполненной работе и ответов на вопросы преподавателя.

Отчет включает в себя следующие разделы: титульный лист с названием работы; цель работы; краткие теоретические сведения; описание результатов лабораторной работы (скриншоты); вывод из работы, включающий в себя описание проделанной работы.

Оценку «отлично» студент получает, если оформление отчета соответсвует установленным требованиям, правильно отвечает на предложенные преподавателем контрольные вопросы, правильно отвечает на дополнительные вопросы по теме лабораторной работы.

Оценку «хорошо» студент получает, если оформление отчета соответсвует установленным требованиям, правильно отвечает на предложенные преподавателем контрольные вопросы.

Оценку «удовлетворительно» студент получает без беседы с преподавателем, если оформление отчета соответсвует установленным требованиям.

Отчет может быть отправлен на доработку в следующих случаях:

- польностью не соовествует установленным требованиям;
- не раскрыта суть работы.

## 6. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ

**ДИСЦИПЛИНЫ**

**6.1. Рекомендуемая литература**

**6.1.1. Перечень основной литературы:**

1. Алексеев, Ю. Е. Введение в информационные технологии и программирование на языке С в среде VS C++. Модуль 1 дисциплины «Информатика» : учебное пособие / Ю. Е. Алексеев. — Москва : Московский государственный технический университет имени Н.Э. Баумана, 2018. — 100 с.

2. Каримов, А. М. Информатика и информационные технологии в профессиональной деятельности : практикум / А. М. Каримов, С. В. Смирнов, Г. Д. Марданов. — Казань : Казанский юридический институт МВД России, 2020. — 120 с.

**6.1.2. Перечень дополнительной литературы:**

1. Липаев, В.В. Качество крупномасштабных программных средств / В.В. Липаев. - М. ; Бе1. Башмакова, Е. И. Информатика и информационные технологии. Технология работы в MS WORD 2016 : учебное пособие / Е. И. Башмакова. — Москва : Ай Пи Ар Медиа, 2020. — 90 с..

2. Мандра, А. Г. Информатика и информационные технологии : лабораторный практикум / А. Г. Мандра, А. В. Попов, А. И. Дьяконов. — 2-е изд. — Самара : Самарский государственный технический университет, ЭБС АСВ, 2020. — 64 с

3. Овчинникова, Е. Н. Информационные технологии. Решение задач в среде программирования VBA : учебное пособие / Е. Н. Овчинникова, С. Ю. Кротова, Т. В. Сарапулова. — Москва : Ай Пи Ар Медиа, 2022. — 101 с.

**6.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине**:

1. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Информационные технологии и программирование».

**6.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:**

1. http://el.ncfu.ru/ – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Цифровая грамотность и обработка данных»
2. http://www.un.org - Сайт ООН Информационно-коммуникационные технологии
3. http://www.intuit.ru – Интернет-Университет Компьютерных технологий.

Информационные справочные системы:

Информационно-справочные и информационно-правовые системы, используемые при изучении дисциплины:

| | |
|---|---|
| 1 | КонсультантПлюс - http://www.consultant.ru/ |

**10. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)**

| | |
|---|---|
| Лекционные занятия | Учебная аудитория с мультимедиа оборудованием. Мультимедийное оборудование: проектор, компьютер, экран настенный.<br>Комплект учебной мебели. |

| | |
|---|---|
| Лабораторные занятия | Лаборатория информационных технологий и систем автоматизированного проектирования с мультимедиа оборудованием. Мультимедийное оборудование: проектор, компьютер, экран настенный. Комплект учебной мебели. |
| Самостоятельная работа | Помещения для самостоятельной работы. Компьютеры с выходом в Интернет и обеспечением доступа в электронную информационно-образовательную среду ИСУ СКФУ. |

Учебные аудитории для проведения учебных занятий, оснащены оборудованием и техническими средствами обучения. Помещения для самостоятельной работы обучающихся, оснащенные компьютерной техникой с возможностью подключения к сети "Интернет" и обеспечением доступа к электронной информационно-образовательной среде. Специализированная мебель и технические средства обучения, служащие для представления учебной информации.

Материально-техническая база обеспечивает проведение всех видов дисциплинарной и междисциплинарной подготовки, лабораторной, научно-исследовательской работы обучающихся (переносной ноутбук, переносной проектор, компьютеры с необходимым программным обеспечением и выходом в интернет).

Помещения для самостоятельной работы обучающихся оснащены компьютерной техникой с возможностью подключения к сети «Интернет» и обеспечением доступа в электронную информационно-образовательную среду образовательной организации.

## 11. Особенности освоения дисциплины (модуля) лицами с ограниченными возможностями здоровья

Обучающимся с ограниченными возможностями здоровья предоставляются специальные учебники, учебные пособия и дидактические материалы, специальные технические средства обучения коллективного и индивидуального пользования, услуги ассистента (помощника), оказывающего обучающимся необходимую техническую помощь, а также услуги сурдопереводчиков и тифлосурдопереводчиков.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья может быть организовано совместно с другими обучающимися, а также в отдельных группах.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья осуществляется с учетом особенностей психофизического развития, индивидуальных возможностей и состояния здоровья.

В целях доступности получения высшего образования по образовательной программе лицами с ограниченными возможностями здоровья при освоении дисциплины (модуля) обеспечивается:

1) для лиц с ограниченными возможностями здоровья по зрению:

- присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочитать и оформить задание, в том числе, записывая под диктовку),
- письменные задания, а также инструкции о порядке их выполнения оформляются увеличенным шрифтом,
- специальные учебники, учебные пособия и дидактические материалы (имеющие крупный шрифт или аудиофайлы),
- индивидуальное равномерное освещение не менее 300 люкс,
- при необходимости студенту для выполнения задания предоставляется увеличивающее устройство;

2) для лиц с ограниченными возможностями здоровья по слуху:

- присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочитать и оформить задание, в том числе, записывая под диктовку),

- обеспечивается наличие звукоусиливающей аппаратуры коллективного пользования, при необходимости обучающемуся предоставляется звукоусиливающая аппаратура индивидуального пользования;

- обеспечивается надлежащими звуковыми средствами воспроизведения информации;

3) для лиц с ограниченными возможностями здоровья, имеющих нарушения опорно-двигательного аппарата (в том числе с тяжелыми нарушениями двигательных функций верхних конечностей или отсутствием верхних конечностей):

- письменные задания выполняются на компьютере со специализированным программным обеспечением или надиктовываются ассистенту;

- по желанию студента задания могут выполняться в устной форме.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

# Методические указания

по выполнению лабораторных работ
по дисциплине
**«ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ»**
для направления подготовки **10.03.01 Информационная безопасность**
направленность (профиль) **Безопасность компьютерных систем**

Пятигорск, 2024

**СОДЕРЖАНИЕ**

**Цель и задачи освоения дисциплины**

Целью изучения дисциплины «Программирование на языках высокого уровня» является ознакомление учащихся с основами современных информационных технологий, тенденциями их развития, получение устойчивых навыков самостоятельной работы на персональном компьютере с применением современных программных средств для получения, хранения и обработки информации, а также получение навыков самостоятельного освоения новых программных средств.

В соответствии с указанной целью при изучении дисциплины «Информационные технологии и программирование» ставятся следующие задачи:

- дать общие характеристики процессов сбора, передачи, обработки и накопления информации;
- познакомить с основами кодирования и сжатия информации;
- дать сведения о технических и программных средствах реализации информационных процессов;
- ознакомить с современными операционными системами и оболочками;
- дать принципы организации, структуры средств систем мультимедиа и компьютерной графики;
- привить навыки работы на современном ПК.

.

**Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы**

| Код, формулировка компетенции | Код, формулировка индикатора | Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов |
|---|---|---|
| ПК-1 Способность выполнять работы по установке, настройке и обслуживанию программных, программно-аппаратных (в том числе криптографических) и технических средств защиты информации | ИД-1 ПК-1 Понимает порядок обслуживания криптографических средств защиты информации.<br>ИД-2 ПК-1. Имеет навыки обслуживать технические средства защиты информации.<br>ИД-3 ПК-1 Владеет навыками эксплуатации программно-аппаратных и технических средств защиты информации. | Использует современные методы обслуживания криптографических средств защиты информации.<br>Применяет на практике навыки обслуживания технических средств защиты информации.<br>Определяет и реализует на практике навыки эксплуатации в программно- аппаратных и технических средств защиты информации. |
| ПК-2 Способность применять программные средства системного, прикладного и специального назначения, инструментальные | ИД-1 ПК-2 Знает методы и средства разработки программного обеспечения.<br>ИД-2 ПК-2 Способен оценивать средства разработки программ.<br>ИД-3 ПК-2 Обладает методами программирования на | эксплуатация и поддержание в рабочем состоянии интегрированных систем безопасности в комплексных системах защиты объектов информатизации |

| | | |
|---|---|---|
| средства, языки и системы программирования для решения профессиональных задач | языках высокого уровня для решения профессиональных задач. | проведение вычислительных экспериментов с использованием стандартных программных средств; определение погрешности измерений при анализе угроз безопасности объектов информатизации |

**Технологическая карта самостоятельной работы студента**

| Коды реализуемых компетенций | Вид деятельности студентов | Средства и технологии оценки | Объем часов, в том числе (акад.) | | |
|---|---|---|---|---|---|
| | | | СРС | Контактная работа с преподавателем | Всего |
| **2 семестр** | | | | | |
| ПК-1 ПК-2 | Самостоятельное изучение литературы и источников | Собеседование | 18,54 | 2,06 | 20,6 |
| ПК-1 ПК-2 | Подготовка к лабораторным занятиям | Защита ЛР | 4,86 | 0,54 | 5,4 |
| ПК-1 ПК-2 | Написание реферата/доклада | Защита доклада | 9 | 1 | 10 |
| | **Итого за 1 семестр** | | **32,4** | **3,6** | **36** |
| | **Итого** | | **32,4** | **3,6** | **36** |

**Содержание самостоятельной работы**

**Тема самостоятельного изучения**: Тема 1. Особенности и области применения языка Python, установка интерпретатора языка и интегрированной среды разработки. Интерфейс и работа с интерпретатором и IDLE. Способы задания значений переменных и задания (изменения) их типов.

**Вид деятельности студентов:** самостоятельное изучение литературы

**Итоговый продукт самостоятельной работы:** конспект

**Средства и технологии оценки:** собеседование

Работа с литературой:

| Рекомендуемые источники информации (№ источника) | | | |
|---|---|---|---|
| Основная | Дополнительная | Методическая | Интернет-ресурсы |
| 1-2 | 1-3 | 1-2 | 1-2 |

**Тема самостоятельного изучения:** Тема 2. Способы создания строк и функции для работы с ними: слияние строк, изменение регистра, получение подстрок, выравнивание.

**Вид деятельности студентов:** самостоятельное изучение литературы

**Итоговый продукт самостоятельной работы:** конспект

**Средства и технологии оценки:** собеседование

Работа с литературой:

| Рекомендуемые источники информации (№ источника) | | | |
|---|---|---|---|
| Основная | Дополнительная | Методическая | Интернет-ресурсы |
| 1-2 | 1-3 | 1-2 | 1-2 |

**Тема самостоятельного изучения:** Тема 3. Основные функции для работы со списками и кортежами в языке Python, особенности и области применения этих конструкций.

**Вид деятельности студентов:** самостоятельное изучение литературы

**Итоговый продукт самостоятельной работы:** конспект
**Средства и технологии оценки:** собеседование
Работа с литературой:

| Рекомендуемые источники информации (№ источника) | | | |
|---|---|---|---|
| Основная | Дополнительная | Методическая | Интернет-ресурсы |
| 1-2 | 1-3 | 1-2 | 1-2 |

**Тема самостоятельного изучения:** Тема 4. Особенности структур словарей и множеств, основные функции для работы с ними, особенности и области их применения.
**Вид деятельности студентов:** самостоятельное изучение литературы
**Итоговый продукт самостоятельной работы:** конспект
**Средства и технологии оценки:** собеседование
Работа с литературой:

| Рекомендуемые источники информации (№ источника) | | | |
|---|---|---|---|
| Основная | Дополнительная | Методическая | Интернет-ресурсы |
| 1-2 | 1-3 | 1-2 | 1-2 |

**Тема самостоятельного изучения:** Тема 5. Способы оформления структуры кода в языке Python. Применение линейных алгоритмов и алгоритмов ветвления и выбора.
**Вид деятельности студентов:** самостоятельное изучение литературы
**Итоговый продукт самостоятельной работы:** конспект
**Средства и технологии оценки:** собеседование
Работа с литературой:

| Рекомендуемые источники информации (№ источника) | | | |
|---|---|---|---|
| Основная | Дополнительная | Методическая | Интернет-ресурсы |
| 1-2 | 1-3 | 1-2 | 1-2 |

**Тема самостоятельного изучения:** Тема 6. Команды создания циклов в языке Python. Различия между циклами while и for и области их применения. Применение команд break и continue.
**Вид деятельности студентов:** самостоятельное изучение литературы
**Итоговый продукт самостоятельной работы:** конспект
**Средства и технологии оценки:** собеседование
Работа с литературой:

| Рекомендуемые источники информации (№ источника) | | | |
|---|---|---|---|
| Основная | Дополнительная | Методическая | Интернет-ресурсы |
| 1-2 | 1-3 | 1-2 | 1-2 |

## 8. Учебно-методическое и информационное обеспечение дисциплины

### 8.1. Перечень основной и дополнительной литературы, необходимой для освоения дисциплины (модуля)

#### 8.1.1. Перечень основной литературы:

1. Алексеев, Ю. Е. Введение в информационные технологии и программирование на языке С в среде VS C++. Модуль 1 дисциплины «Информатика» : учебное пособие / Ю. Е. Алексеев. — Москва : Московский государственный технический университет имени Н.Э. Баумана, 2018. — 100 с.

2. Каримов, А. М. Информатика и информационные технологии в профессиональной деятельности : практикум / А. М. Каримов, С. В. Смирнов, Г. Д.

Марданов. — Казань : Казанский юридический институт МВД России, 2020. — 120 с.

**8.1.2. Перечень дополнительной литературы:**

1. Башмакова, Е. И. Информатика и информационные технологии. Технология работы в MS WORD 2016 : учебное пособие / Е. И. Башмакова. — Москва : Ай Пи Ар Медиа, 2020. — 90 с..
2. Мандра, А. Г. Информатика и информационные технологии : лабораторный практикум / А. Г. Мандра, А. В. Попов, А. И. Дьяконов. — 2-е изд. — Самара : Самарский государственный технический университет, ЭБС АСВ, 2020. — 64 с
3. Овчинникова, Е. Н. Информационные технологии. Решение задач в среде программирования VBA : учебное пособие / Е. Н. Овчинникова, С. Ю. Кротова, Т. В. Сарапулова. — Москва : Ай Пи Ар Медиа, 2022. — 101 с.

**8.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине (модулю)**

1. Методические рекомендации по выполнению лабораторных работ по дисциплине " Информационные технологии и программирование "
2. Методические рекомендации по организации самостоятельной работы студентов по дисциплине " Информационные технологии и программирование "

**8.3. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины (модуля)**

1. http://el.ncfu.ru/ – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Цифровая грамотность и обработка данных»
2. http://www.un.org - Сайт ООН Информационно-коммуникационные технологии
3. http://www.intuit.ru – Интернет-Университет Компьютерных технологий.

**9. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем**

При чтении лекций используется компьютерная техника, демонстрации презентационных мультимедийных материалов. На семинарских и практических занятиях студенты представляют презентации, подготовленные ими в часы самостоятельной работы.

Информационные справочные системы:

Информационно-справочные и информационно-правовые системы, используемые при изучении дисциплины:

| | |
|---|---|
| 1 | КонсультантПлюс - http://www.consultant.ru/ |

**10. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)**

| | |
|---|---|
| Лекционные занятия | Учебная аудитория с мультимедиа оборудованием. Мультимедийное оборудование: проектор, компьютер, экран настенный. Комплект учебной мебели. |
| Практические занятия | Лаборатория информационных технологий и систем автоматизированного проектирования с мультимедиа оборудованием. Мультимедийное оборудование: проектор, компьютер, экран настенный. Комплект учебной мебели. |

| | |
|---|---|
| Самостоятельная работа | Помещения для самостоятельной работы. Компьютеры с выходом в Интернет и обеспечением доступа в электронную информационно-образовательную среду ИСУ СКФУ. |

Учебные аудитории для проведения учебных занятий, оснащены оборудованием и техническими средствами обучения. Помещения для самостоятельной работы обучающихся, оснащенные компьютерной техникой с возможностью подключения к сети "Интернет" и обеспечением доступа к электронной информационно-образовательной среде. Специализированная мебель и технические средства обучения, служащие для представления учебной информации.

Материально-техническая база обеспечивает проведение всех видов дисциплинарной и междисциплинарной подготовки, лабораторной, научно-исследовательской работы обучающихся (переносной ноутбук, переносной проектор, компьютеры с необходимым программным обеспечением и выходом в интернет).

Помещения для самостоятельной работы обучающихся оснащены компьютерной техникой с возможностью подключения к сети «Интернет» и обеспечением доступа в электронную информационно-образовательную среду образовательной организации.

**11. Особенности освоения дисциплины (модуля) лицами с ограниченными возможностями здоровья**

Обучающимся с ограниченными возможностями здоровья предоставляются специальные учебники, учебные пособия и дидактические материалы, специальные технические средства обучения коллективного и индивидуального пользования, услуги ассистента (помощника), оказывающего обучающимся необходимую техническую помощь, а также услуги сурдопереводчиков и тифлосурдопереводчиков.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья может быть организовано совместно с другими обучающимися, а также в отдельных группах.

Освоение дисциплины (модуля) обучающимися с ограниченными возможностями здоровья осуществляется с учетом особенностей психофизического развития, индивидуальных возможностей и состояния здоровья.

В целях доступности получения высшего образования по образовательной программе лицами с ограниченными возможностями здоровья при освоении дисциплины (модуля) обеспечивается:

1) для лиц с ограниченными возможностями здоровья по зрению:

- присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочитать и оформить задание, в том числе, записывая под диктовку),
- письменные задания, а также инструкции о порядке их выполнения оформляются увеличенным шрифтом,
- специальные учебники, учебные пособия и дидактические материалы (имеющие крупный шрифт или аудиофайлы),
- индивидуальное равномерное освещение не менее 300 люкс,
- при необходимости студенту для выполнения задания предоставляется увеличивающее устройство;

2) для лиц с ограниченными возможностями здоровья по слуху:

- присутствие ассистента, оказывающий студенту необходимую техническую помощь с учетом индивидуальных особенностей (помогает занять рабочее место, передвигаться, прочитать и оформить задание, в том числе, записывая под диктовку),
- обеспечивается наличие звукоусиливающей аппаратуры коллективного

пользования, при необходимости обучающемуся предоставляется звукоусиливающая аппаратура индивидуального пользования;

- обеспечивается надлежащими звуковыми средствами воспроизведения информации;

3) для лиц с ограниченными возможностями здоровья, имеющих нарушения опорно-двигательного аппарата (в том числе с тяжелыми нарушениями двигательных функций верхних конечностей или отсутствием верхних конечностей):

- письменные задания выполняются на компьютере со специализированным программным обеспечением или надиктовываются ассистенту;

- по желанию студента задания могут выполняться в устной форме.