Қазақстан Республикасының Оқу-ағарту министрлігі
Министерство просвещения Республики Казахстан
«Тұран» университетінің колледжі
Колледж университета «Туран»

ДАЙЫНДАҒАН:
РАЗРАБОТАНО:
Николаенкова Т.Е , информатика және ақпараттық пәндер оқытушысы
Николаенкова Т.Е., преподаватель информатики и информационных дисциплин

ҚАРАСТЫЛЫҒАН
РАССМОТРЕНО
Жалпы білім беру пәндердің ЦӘК отырысында
на заседании ЦМК технических дисциплин
Хаттама / Протокол № 4 от 23.11. 2024 ж./г
ЦӘК жетекшісі /Председатель ЦМК: _____ Зординова З.М.

ҰСЫНЫЛҒАН
РЕКОМЕНДОВАНО
әдістемелік кеңес отырысында
на заседании методического совета
« 12 » « 01 » 2025 ж/г. хаттама /протокол № 5

БЕКІТЕМІН
УТВЕРЖДАЮ
«Тұран» университеті колледжінің директоры
Директор колледжа университета «Туран»
_____ Баймагамбетова Х.В.

ЯЗЫК ПРОГРАММИРОВАНИЯ PYTHON

Учебно-методическое пособие
Для обучающихся 1 курса
Специальности 06130100 «Программное обеспечение»
Колледжа «Туран»

Алматы, 2025

Рецензент: Мамырова А.К., кандидат технических наук, ассоциированный профессор Высшей школы информационных технологий

Язык программирования Python. Учебно-методическое пособие для обучающихся 1 курса специальности «Программное обеспечение» колледжа «Туран» - Алматы, 2025 - 51с.

Учебно-методическое пособие предназначено для использования в рамках факультативных занятий для обучающихся 1 курса специальности 06130100 «Программное обеспечение» колледжа «Туран».

Пособие содержит теоретический материал по основам программирования на языке программирования Python, перечень практических работ, с примерами и контрольными вопросами по каждому тематическому разделу.

СОДЕРЖАНИЕ

Пояснительная записка

Учебно-методическое пособие содержит теоретический материал по основам программирования на языке *Python*, перечень практических работ с рекомендациями по их выполнению, примерами и контрольными вопросами по каждому тематическому разделу.

В пособии представлены следующие тематические блоки:

- основные сведения о языке программирования Python;
- переменные, выражения и инструкции;
- простые типы данных и операции над ними;
- условная инструкция if для реализации ветвлений
- списки в Python
- инструкции цикла в Python
- строки и обработка текстовой информации;
- функции
- кортежи, словари и множества.

Цель пособия – рассказать об основах программирования для слушателей с минимальным знанием информатики. Данный курс позволяет научиться проектировать и разрабатывать приложения, используя базовые возможности языка программирования Python.

Изучение материала направлено на развитие навыков и умений, обеспечивающих возможность разработки алгоритмов и программ с использованием основных конструкций языка программирования. В связи с этим значительное внимание уделено особенностям языка Python на примерах конкретных задач из различных разделов программирования.

Предлагаемое учебно-методическое пособие предназначено для использования в рамках факультативных занятий для начального обучения программированию. Освоение практикума предполагает выполнение обучающимся ряда практических работ для формирования начальных практических навыков программирования и закрепления теоретического материала. Особенностью пособия является во многом самостоятельная деятельность обучающегося при наличии необходимого контроля со стороны преподавателя и дифференцированный подход к распределению задач.

I. ЗНАКОМСТВО С ЯЗЫКОМ ПРОГРАММИРОВАНИЯ PYTHON

1.1. Основные сведения о Python

Python – это современный и мощный высокоуровневый язык программирования, разработанный голландским программистом Гвидо ван Россумом и представленный в 1991 году. Язык получил своё название в честь популярного комедийного телешоу «Воздушный цирк Монти Пайтона», которое транслировалось на BBC. Это добавляет элемент юмора в примеры кода на *Python*.

Python был разработан с учётом влияния таких языков, как *Java, C* и *C++* (по словам самого создателя, он выбрал наиболее непротиворечивые конструкции из языка *C*, чтобы не вызвать у С-программистов отрицательного отношения к новому языку). В свою очередь, *Python* оказал влияние на язык *Ruby*.

Официальный сайт *Python* – http://python.org.

Python можно охарактеризовать как:

- интерпретируемый язык: исходный код не компилируется в машинный код, а исполняется с помощью специальной программы-интерпретатора;
- интерактивный язык: возможность писать код непосредственно в интерпретаторе, вводя новые команды по мере выполнения предыдущих;
- объектно-ориентированный язык: поддерживает принципы объектно-ориентированного программирования, что включает инкапсуляцию кода в объекты – особые структуры данных.

Особенности языка *Python*:

- **Легкость в изучении**: *Python* обладает ограниченным набором ключевых слов, простой структурой и чётко определённым синтаксисом, что позволяет освоить основы языка за сравнительно короткий период.
- **Читаемость кода**: Блоки кода в *Python* выделяются с помощью отступов, а использование ключевых слов, основанных на английском языке, делает код более понятным и лёгким для восприятия.
- **Удобство в обслуживании кода**: Код на *Python* прост в поддержке и модификации благодаря своей ясной структуре.
- **Широкая кросс-платформенность**: *Python* имеет обширную библиотеку, совместимую с разными операционными системами.
- **Интерактивный режим**: Возможность тестировать отдельные участки кода в процессе работы, что значительно ускоряет разработку.
- **Портативность**: *Python* легко запускается на различных платформах и сохраняет идентичный интерфейс, что упрощает переносимость приложений.
- **Расширяемость**: *Python* позволяет интегрировать низкоуровневые модули, написанные на других языках программирования, для решения более сложных задач.

- **Работа с базами данных**: В стандартной библиотеке Python доступны модули для работы с различными базами данных.
- **Создание GUI**: *Python* поддерживает создание графических пользовательских интерфейсов (GUI) с широкими возможностями для настройки и персонализации.

Недостатки *Python*

Скорость выполнения программ на *Pyth*on несколько ниже, чем у компилируемых языков, таких как *C* или *C++*, поскольку *Python* сначала транслирует исходный код в байт-код, а затем интерпретирует его. Байт-код является платформонезависимым, что даёт преимущество в переносимости программ. Однако это также ведёт к некоторому снижению производительности при выполнении, компенсированному ускорением разработки.

Кто использует Python сегодня?

- **Google** поддерживает создателя Python и активно использует язык для разработки поисковых систем;
- **YouTube** использует Python для различных сервисов;
- **Yandex** применяет Python в ряде своих сервисов;
- Программа **BitTorrent** написана с использованием Python;

- Веб-фреймворк **App Engine** от Google использует *Python* в качестве основного языка для разработки приложений;
- Компании **Intel**, **Cisco**, **HP** и **IBM** используют *Python* для тестирования аппаратного обеспечения;
- **NASA** применяет *Python* для научных вычислений и обработки данных.

1.2 Пример написания простейшей программы

Установка Python вместе с IDLE

1. Перейдите на официальный сайт:
☞ https://www.python.org/downloads/
2. Скачайте установщик для Windows.
3. Запустите установочный файл и **обязательно** поставьте галочку ✓ **"Add Python to PATH"**.
4. В процессе установки убедитесь, что отмечена опция **"Install IDLE"**.
5. После завершения установки IDLE можно найти в **Пуск → IDLE (Python x.x.x)**.

После загрузки и установки Python, необходимо открыть IDLE — среду разработки, которая поставляется вместе с дистрибутивом Python (см. рис. 1.1).

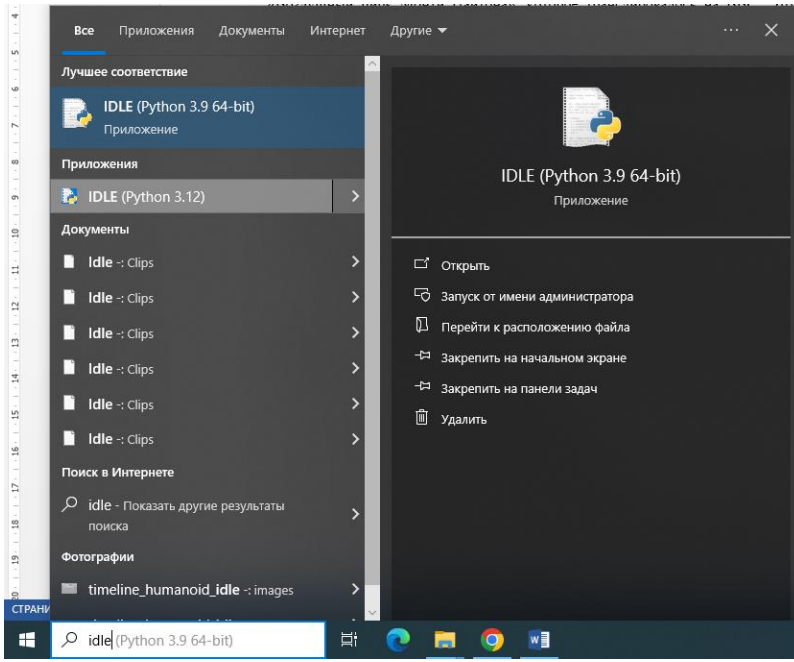


Рис. 1.1 Запуск IDLE

После запуска IDLE (который изначально работает в интерактивном режиме), можно приступить к написанию первой программы. Рассмотрим создание программы, которая выводит на экран приветственное сообщение «Hello, students! :))».

Для этого используется инструкция `print("Hello, students!:))")`. Код вводится в IDLE, и результат выводится сразу после нажатия клавиши Enter. Пример работы программы показан на рис. 1.2.

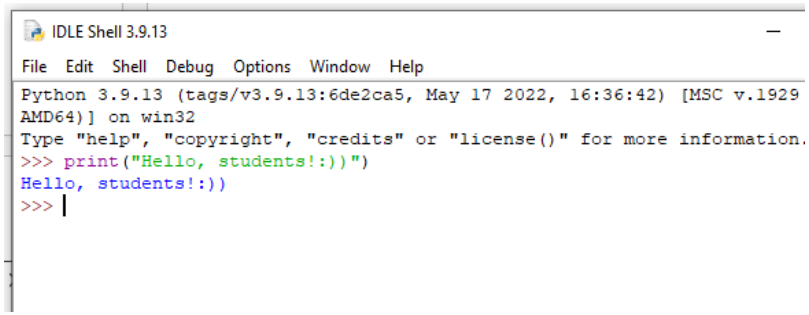


Рис. 1.2 Вывод сообщения

Интерактивный режим удобен для быстрого тестирования и экспериментов, но обладает значительными ограничениями по функционалу. Для полноценной работы над программой, обычно создается код в сценарном режиме, который затем сохраняется в файл, называемый **скриптом** (script), и запускается после его сохранения.

Чтобы перейти в сценарный режим и создать новый файл в IDLE, необходимо выбрать **File → New File** или нажать сочетание клавиш **Ctrl + N** (см. рис. 1.3).

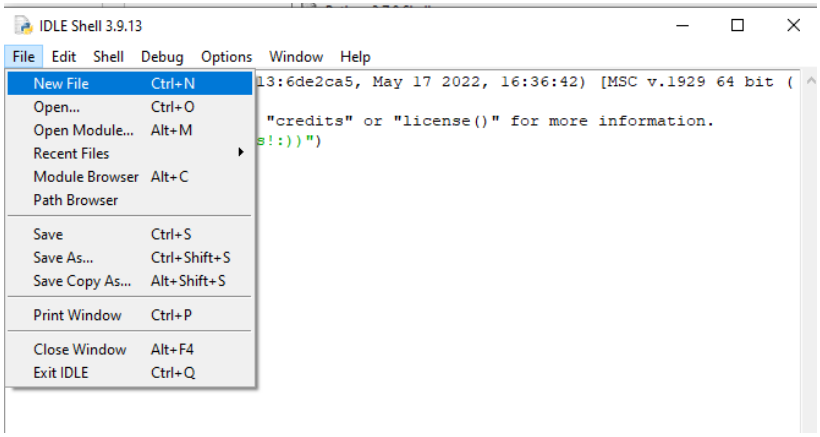


Рис. 1.3 Создание нового окна

В качестве примера создадим программу, которая запрашивает у пользователя его имя и приветствует его, обращаясь по имени. Первая строка программы задаёт вопрос «Как вас зовут?», ожидает, пока пользователь введет ответ, и сохраняет введённое имя в переменную name.

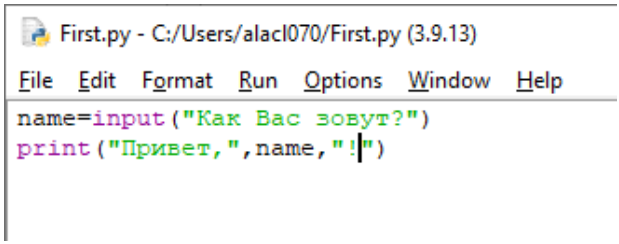


Рис. 1.4 Код программы

Для того чтобы запустить программу, необходимо нажать клавишу **F5** или выбрать в меню IDLE пункт **Run → Run Module**. Программа предложит сохранить файл, и после этого она будет выполнена. Исходные файлы Python имеют расширение *.py. Результат работы программы можно увидеть на рис. 1.5.

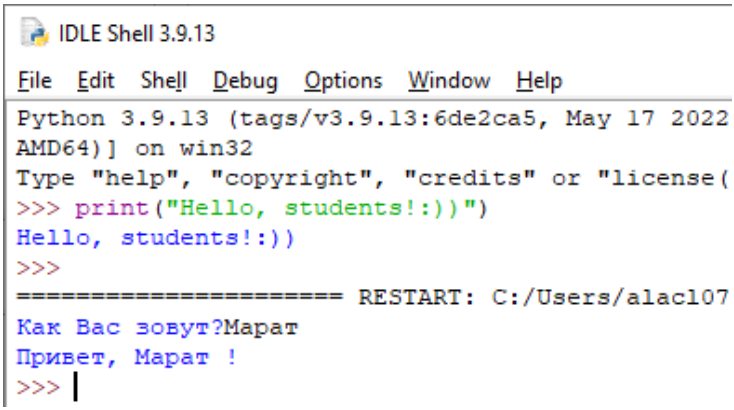


Рис. 1.5 Результат запуска программы

1.3. Комментарии в Python

Комментарии — это строки в коде, которые игнорируются интерпретатором Python и не выполняются как часть программы. Они служат для:

- Описания работы кода, что упрощает его понимание.
- Улучшения читаемости и поддержки кода в будущем.
- Отключения части кода без его удаления (например, для отладки).

Виды комментариев в Python

Комменнтарии бывают двух видов однострочные и многострочные.Однострочные комментарии начинаются с символа # и продолжаются до конца строки. Они используются для кратких пояснений.

Многострочные используются для больших пояснений к коду, заключаются в тройные кавычки (''' или """). Часто применяются для документации функций и классов. (см.рис.1.6)

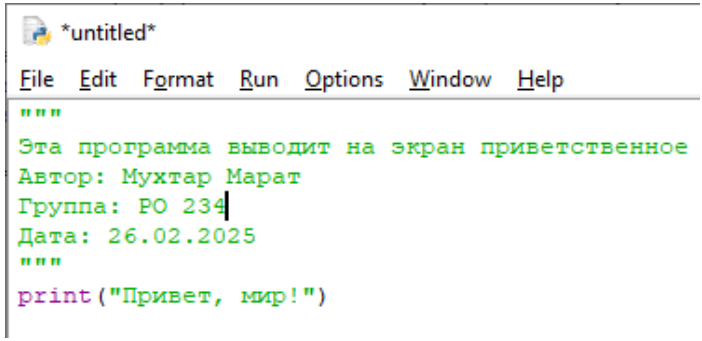


Рис. 1.6 Пример многострочного комментария

Практическая работа 1
«Знакомство с языком программирования Python»

Цель работы: Научиться создавать, редактировать, сохранять и запускать программы на языке *Python*.

Задание:

1. Запустить встроенную среду разработки Python;
2. Протестировать программный код, приведенный в предидущем разделе в интерактивном режиме и в режиме создания скрипта.
3. Изучить основные горячие клавиши, которые ускоряют процесс написания кода в сценарном режиме.
4. Создать программу, которая выводит на экран кличку Вашего любимого питомца.

Контрольные вопросы:

1. Что такое язык программирования?
2. Что такое программа?
3. Почему можно рассматривать Python как интерпретируемый, интерактивный и объектно-ориентированный язык программирования?
4. Назовите особенности языка Python.
5. Какие факторы влияют на скорость выполнения программ на Python?
6. Для чего предназначена среда разработки IDLE?
7. В чем разница между интерактивным и сценарным режимами работы в IDLE?
8. Почему важно использовать комментарии в коде?
9. Назовите виды комментариев в Python и их отличие?
10. Каким образом можно вывести текст на экран в Python?

II. ПЕРЕМЕННЫЕ, ВЫРАЖЕНИЯ И ИНСТРУКЦИИ

2.1. Переменные и инструкция присваивания

Переменная — это имя, которое ссылается на значение. В *Python* переменная создаётся автоматически при присвоении значения, и её тип определяется динамически.

```
x = 10  # Переменная x содержит целое число 10
name = "Марат"  # Переменная name содержит строку "Марат"
```

Хорошей практикой считается давать переменным осмысленные имена, которые четко отражают их назначение. Длина имени переменной не ограничена, оно может включать латинские буквы и цифры, но не должно начинаться с цифры. Допускается использование заглавных букв, однако в сообществе разработчиков принято писать имена переменных строчными буквами.

Если вы присвоите переменной недопустимое имя, то при выполнении программы увидите синтаксическую ошибку: SyntaxError: invalid syntax

Еще одним правилом именования переменных является недопустимость задания имен переменных совпадающих с зарезерверированными словами. Ниже приведен список зарезервированных слов.

Инструкция присваивания обозначается символом “=”, создает новую переменную и задает ей значение:

```
>>> message = ' Это сообщение '
>>> n = 33
>>> pi = 3.1415
```

В этом примере три инструкции присваивания. Первая присваивает строку новой переменной с именем message; вторая задает переменной n целочисленное значение 17; третья присваивает приближенное значение π переменной pi.

```
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
```

Рис 2.1 Зарезервированные слова Python

2.2. Функции PRINT() и INPUT()

Функция **print()** предназначена для вывода информации на экран. Это базовая функция, используемая для отображения одного или нескольких значений, по умолчанию выводящихся в консоль. У функции **print()** имеются дополнительные параметры:

- **sep** – строка или символ, вставляемый между выводимыми значениями. По умолчанию используется пробел.
- **end** – строка, добавляемая после последнего выводимого значения. По умолчанию это символ новой строки (\n).

Дополнительные параметры указываются в конце списка аргументов (см. рис. 2.1).

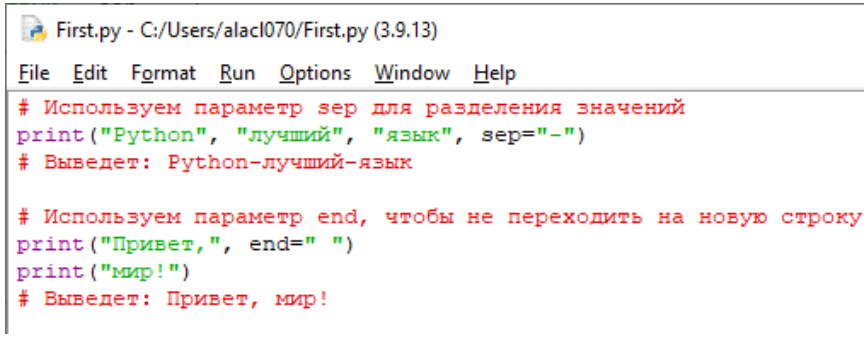


Рис.2.2 Пример использования дополнительных параметров

input() — это стандартная функция, предназначенная для ввода данных пользователем. Она может принимать необязательный аргумент — текстовое приглашение, которое отображается перед вводом. Введённые данные возвращаются в виде строки.

Основные правила работы с input():

- Строка-приглашение **не включает автоматический перенос на новую строку**, поэтому важно заранее продумать удобную форму запроса.
- Часто используется для создания **паузы перед закрытием программы**, позволяя пользователю увидеть результаты её выполнения.

Синтаксис функции input() представлен на рисунке 2.2.

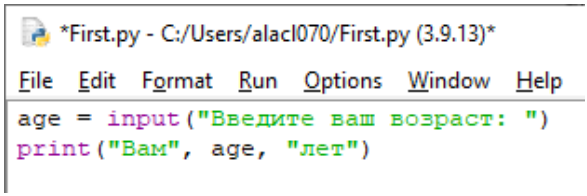


Рис 2.3 Синтаксис функции input()

Практическая работа 2

«Использование функций для ввода и вывода» информации

Цель работы: Ознакомиться с функциями input() и print(), научиться организовывать ввод и вывод данных в *Python*, а также использовать параметры этих функций для удобного форматирования информации.

Задание:

- Напишите программу, которая запрашивает у пользователя три слова и выводит их через разделитель " * ". Используйте параметр sep в print().
- Создайте программу, которая запрашивает у пользователя:

  Имя
  Город проживания
  Любимый цвет
  После ввода программа должна вывести информацию в виде анкеты.

- Напишите программу, которая запрашивает у пользователя его девиз по жизни, затем выводит его на экран. После этого программа должна ждать нажатия клавиши Enter перед завершением.

Контрольные вопросы:

1. Какими способами отображаются строки в Python?
2. Зачем в Python используют тройные кавычки (апострофы)?
3. Какая функция используется для ввода данных в Python?
4. Какая функция используется для вывода данных в Python?
5. Как вывести несколько значений через print() в одной строке?
6. Как задать разделитель между выводимыми элементами в print()?

III. ПРОСТЫЕ ТИПЫ ДАННЫХ И ОПЕРАЦИИ НАД НИМИ

3.1. Простые типы данных

В памяти компьютера информация может храниться в различных форматах. В Python существуют стандартные типы данных, среди которых: число (Number), строка (String), список (List), кортеж (Tuple), словарь (Dictionary) и множество (Set).

Числовой тип данных предназначен для хранения числовых значений и относится к неизменяемым типам. Это означает, что при изменении числового значения создаётся новый объект в памяти, а старый удаляется.

Python строки могут быть объявлены несколькими способами:

1. с использованием одинарных кавычек ('...');
2. с использованием двойных кавычек ("...");
3. с использованием тройных кавычек ('''...''' или """...""").

Строки, заключённые в тройные кавычки, обладают особенностью — они могут занимать несколько строк в коде и при этом сохраняют исходное форматирование при выводе.

Списки, кортежи, словари и множества представляют собой структуры данных, хранящие последовательности значений различных типов. Списки, множества и словари представляют собой изменяемые последовательности.

Python использует динамическую типизацию, что означает, что переменные не привязаны к определённому типу данных. Каждая переменная является ссылкой на объект, а её тип определяется тем, на что эта ссылка указывает. В процессе выполнения программы переменная может изменять свой тип, если ей присваивается значение другого типа.

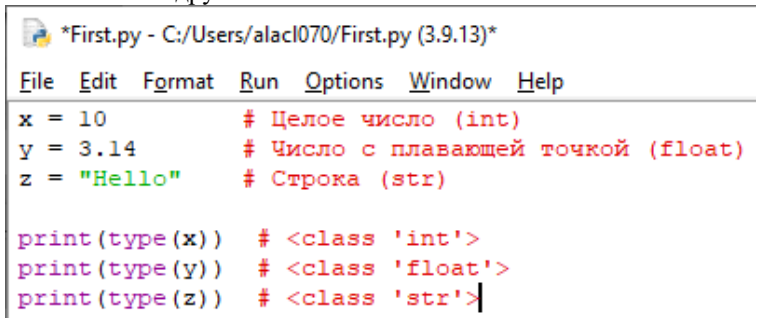


Рис 3.1 Типы переменных

3.2. Операторы и операнды в Python

Операторы — это специальные символы или ключевые слова, которые выполняют операции над значениями.

Операнды — это значения, над которыми выполняется операция.

Арифметические операторы - используются для выполнения математических вычислений. Описание пример и результаты арифметических операторов приседены в таблице 3.1.

Таблица 3.1

Арифметичекие операторы

Оператор	Описание	Пример	Результат
+	Сложение	5 + 3	8
-	Вычитание	10 - 4	6
*	Умножение	6 * 2	12
/	Деление (с п лавающей точкой)	7 / 2	3.5
//	Целочисленное деление	7 // 2	3
%	Остаток от деления	7 % 2	1
**	Возведение в степень	2 ** 3	8

Операторы сравнения - возвращают True или False, используются для сравнения значений**.** Операторы сравнения часто используются в логических выражениях при реализации разветвляющихся алгоритмов**.** Описание пример и результаты операторов сравнения приседены в таблице 3.2.

Таблица 3.2

Операторы сравнения

Оператор	Описание	Пример	Результат
==	Равно	5 == 5	True
!=	Не равно	5 != 3	True
>	Больше	5 > 3	True
<	Меньше	3 < 5	True
>=	Больше или равно	5 >= 5	True
<=	Меньше или равно	3 <= 5	True

Логические операторы - используются для работы с булевыми значениями (True или False).

Таблица 3.3

Логические операторы

Оператор	Описание	Пример	Результат
and	Логическое «И» (True, если оба True)	True and False	False
or	Логическое «ИЛИ» (True, если хотя бы одно True)	True or False	True
not	Логическое «НЕ» (инвертирует значение)	not True	False

Операторы присваивания - используются для присвоения значений переменным**.**

Таблица 3.4.
Операторы присваивания

Оператор	Эквивалент	Пример	Результат
=	x = 5	x = 5	x станет 5
+=	x = x + 3	x += 3	Увеличит x на 3
-=	x = x - 2	x -= 2	Уменьшит x на 2
*=	x = x * 4	x *= 4	Умножит x на 4
/=	x = x / 2	x /= 2	Разделит x на 2
//=	x = x // 3	x //= 3	Целочисленное деление
%=	x = x % 5	x %= 5	Остаток от деления
**=	x = x ** 2	x **= 2	Возведение в степень

Практическая работа 3
«Простые типы данных и операции над ними»

Цель работы: Закрепить знания о простых типах данных в Python (числовые, строковые, логические) и научиться применять операции над ними.

Задание:

1. Напишите программу, которая запрашивает два числа у пользователя и выводит их сумму, разность, произведение и частное.
2. Запросите у пользователя два числа и выведите остаток от их деления и результат возведения первого числа в степень второго.
3. Запросите у пользователя два слова и объедините их в одну строку с пробелом между ними.
4. Запросите у пользователя слово и число, затем выведите это слово, повторённое указанное количество раз.
5. Запросите у пользователя число и определите, является ли оно четным или нечетным.
6. Запросите у пользователя слово и выведите его первый и последний символы.

Проект:

"Простой калькулятор". Создать программу, которая выполняет основные арифметические операции (сложение, вычитание, умножение, деление) с двумя числами, введенными пользователем.

Этапы выполнения проекта:

1. **Ввод чисел:**

   Программа должна запросить у пользователя два числа.
   Преобразовать введенные строки в числа (целые или дробные).

2. **Выбор операции:**

   Программа должна запросить у пользователя, какую операцию он хочет выполнить (+, -, *, /).

3. **Выполнение операции:**

   В зависимости от выбранной операции выполнить соответствующее арифметическое действие.
   Обработать ошибку деления на ноль.

4. **Вывод результата:**

   Вывести результат операции на экран.

Контрольные вопросы:

1. Какие стандартные типы данных существуют в Python?
2. Что такое динамическая типизация в Python?
3. Как определить тип переменной в Python?
4. Какие операции можно выполнять с целыми (int) и вещественными (float) числами?
5. Как выполняется целочисленное деление и чем оно отличается от обычного деления?
6. Какой оператор необходимо использовать для возведения в степень?
7. Как получить остаток от деления в Python?
8. Какие операторы относятся к логическим?
9. Какие логические (bool) значения существуют в Python?
10. Что означает операция инкремента?

IV. УСЛОВНАЯ ИНСТРУКЦИЯ IF ДЛЯ РЕАЛИЗАЦИИ ВЕТВЛЕНИЙ

4.1. Базовый синтаксис условных конструкций

Разветвляющимся называется такой алгоритм, в котором выбирается один из нескольких возможных путей (вариантов) вычислительного процесса. Очень часто для реализации разветвляющихся алгоритмов используется инструкция if, где в качестве условия могут выступать логические выражения.

Логическими (булевыми) выражениями называют такие выражения, которые могут принимать одно из двух значений: истину или ложь. Операнды логических операторов должны быть логическими выражениями. В языке Python любое ненулевое число трактуется как «истинное».

В общем виде синтаксически неполный условный оператор выглядит следующим образом (рис 4.1.):

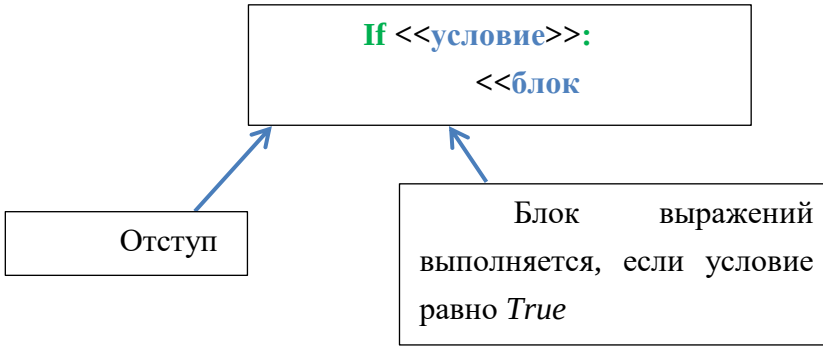


Рис. 4.1 Синтаксис условной конструкции

На рис. 4.2 и рис. 4.3 представлена блок-схема и программный код для проверки положительности числа x с выводом соответствующего сообщения.

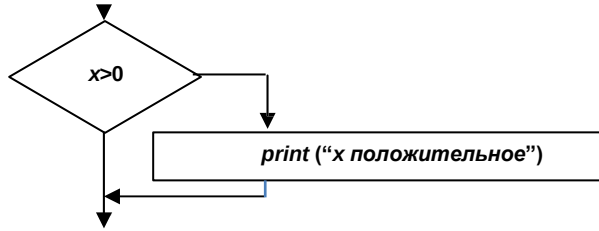
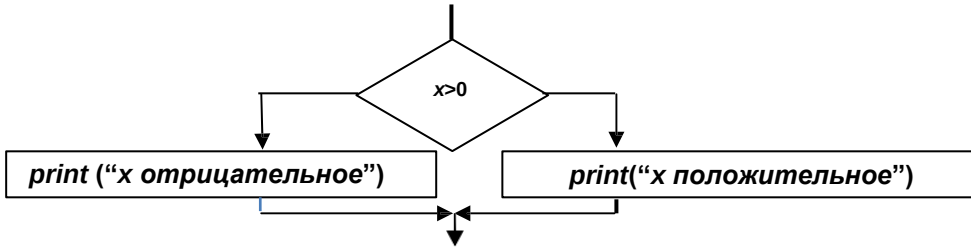


Рис. 4.2 Блок-схема неполного условного оператора

```
x = float(input("Введите число: "))
if x > 0:
      print("Число положительное")
```

Рис. 4.3 Программный код неполного условного оператора на языке Python

Полный условный оператор предполагает два направления выполнения, и условие определяет, какое из них выполняется. (рис 4.4)




```
x = float(input("Введите число: "))
if x > 0:
   print("Число положительное.")
else
   print("Число отрицательное.")
```

Рис. 4.4 Блок-схема полного условного оператора и программный код на языке Python

4.2. Вложенные конструкции if

Конструкция if в свою очередь сама может иметь вложенные конструкции if. Пример вложенной конструкции if приведен на рис. 4.5

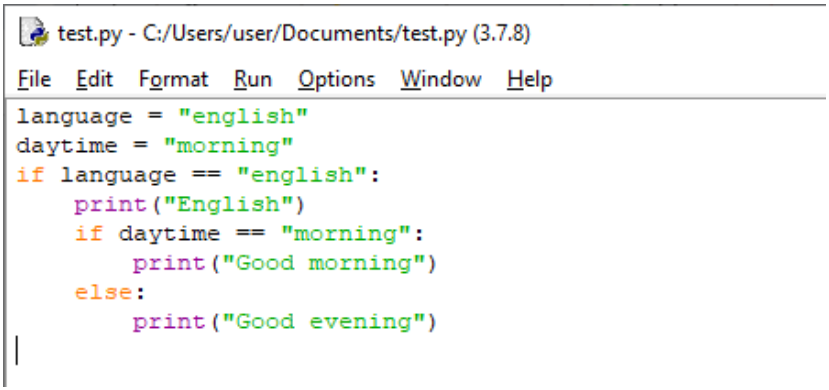


Рис. 4.5 Вложенная конструкция if

Стоит учитывать, что вложенные выражения if также должны начинаться с отступов, а инструкции во вложенных конструкциях также должны иметь отступы. Отступы, расставленные не должным образом, могут изменить логику программы.

Можно «объединить» первое служебное слово else и следующее за ним слово if и записать слово elif. Условное выражение может включать множество проверок. Общий синтаксис у него представлен на рисунке 4.6:

```
If <<условие>>:
        <<блок выражений>>
elif <<условие>>:
        <<блок выражений>>
else:
  <<блок выражений>>
```

Рис 4.6 Общий синтаксис оператора elif

Блок выражений, относящийся к else, выполняется, когда все вышестоящие условия вернули False. Использование служебного слова elif позволяет упростить запись и избежать чрезмерных отступов.

Практическая работа 4 «Условная инструкция if»

Цель работы: Научиться использовать логические операторы (and, or, not) для создания сложных условий. Научиться писать программы, которые принимают решения в зависимости от заданных условий

Задания:

1. Напишите программу, которая запрашивает у пользователя число и выводит, является ли оно положительным, отрицательным или нулевым.
2. Создайте программу, которая запрашивает у пользователя возраст и определяет, является ли он совершеннолетним (например, старше 18 лет).
3. Напишите программу, которая запрашивает у пользователя два числа и выводит наибольшее из них.
4. Напишите программу, которая запрашивает у пользователя число и определяет четное оно или нет.
5. Создайте программу, которая запрашивает у пользователя коэффициенты квадратного уравнения и выводит его корни (с учетом различных случаев: два корня, один корень, нет корней).

Проект:

Разработать программу, которая определяет, является ли введенное пользователем число четным или нечетным.

Этапы выполнения:

1. Запросить у пользователя число.
2. Преобразовать введенную строку в целое число.
3. Использовать условный оператор `if` для проверки остатка от деления числа на 2.
4. Вывести соответствующее сообщение ("Число четное" или "Число нечетное").

Контрольные вопросы:

1. Что такое условная инструкция if и для чего она используется?
2. Как выглядит синтаксис инструкции if?
3. Что такое условие в инструкции if?
4. Что делает ключевое слово else?
5. Что делает ключевое слово elif (или else if)?
6. Какие логические операторы можно использовать в условиях?

V. СПИСКИ В PYTHON

5.1. Понятие и создание списка

В Python списки (lists) — это упорядоченные изменяемые коллекции элементов. Они являются одним из самых гибких и часто используемых типов данных в Python.

Есть несколько способов создать список:

- **Пустой список:**

my_list = []

- **Список с элементами:**

my_list = [1, 2, 3, 'hello', True]

- **Использование функции list():**

my_list = list((1, 2, 3)) # Создание списка из кортежа

Пример создания списка приведен на рисунке 5.1.

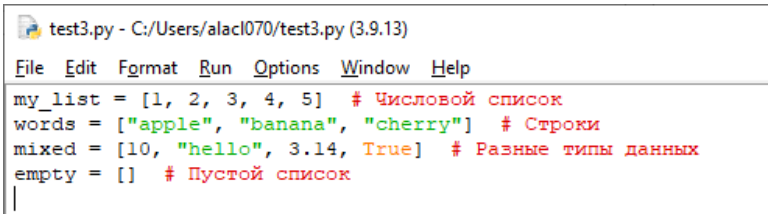


Рис. 5.1 Создание списка

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списка. Индексы начинаются с нуля. То есть первый элемент будет иметь индекс 0, второй элемент - индекс 1 и так далее. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.

Python работает со списками в памяти следующим образом: Переменная e содержит адрес списка (id6). Каждый элемент списка является указателем (хранит адрес) другого объекта (в данном случае вещественных чисел). Схематично работа со списками представлена на рисунке 5.2

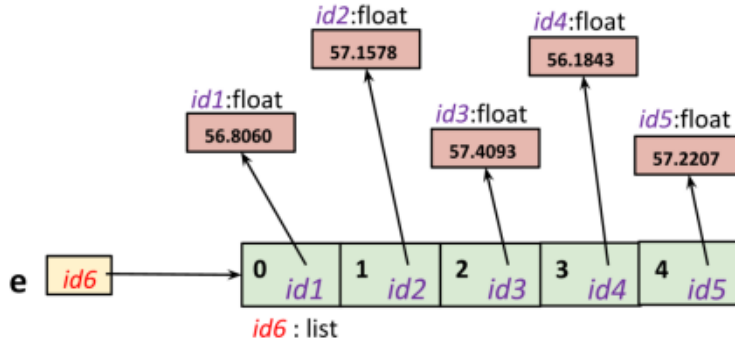


Рис 5.2 Представление списка в памяти

5.2. Методы и функции по работе со списками

Для управления элементами списки имеют целый ряд методов.

В таблице 5.1 приведены основные функции и методы для работы со списками в применяемые Python, их описания и примеры использования.

Таблица 5.1

Основные операции списков

Операция	Описание	Пример
Доступ к элементам	list[index]	first_item = my_list[0]
Изменение элемента	list[index] = new_value	my_list[1] = 5
Добавление элемента в конец	list.append(item)	my_list.append(4)
Вставка элемента	list.insert(index, item)	my_list.insert(2, 7)
Удаление элемента по значению	list.remove(item)	my_list.remove(2)
Удаление элемента по индексу	del list[index]	del my_list[0]
Удаление и возврат последнего элемента	list.pop()	last_item = my_list.pop()
Удаление и возврат элемента по индексу	list.pop(index)	second_item = my_list.pop(1)

Операция	Описание	Пример
Очистка списка	list.clear()	my_list.clear()
Длина списка	len(list)	length = len(my_list)
Срезы (slices)	list[start:end]	sub_list = my_list[1:3]
Сортировка списка	list.sort()	my_list.sort()
Переворачивание списка	list.reverse()	my_list.reverse()
Копирование списка	list.copy()	new_list = my_list.copy()
Расширение списка	list.extend(iterable)	my_list.extend([4, 5, 6])
Поиск индекса элемента	list.index(item)	index = my_list.index(3)
Подсчет количества вхождений элемента	list.count(item)	count = my_list.count(2)

Практическая работа 5
«Списки в Python»

Цель работы: Научиться использовать логические операторы (and, or, not) для создания сложных условий. Научиться писать программы, которые принимают решения в зависимости от заданных условий

Задания:

Задан список L = [3, 6, 7, 4, -5, 4, 3, -1]

1. Определите сумму элементов списка L. ЕСЛИ сумма превышает значение 2, ТО вывести на экран число элементов списка.
2. Определить разность между минимальным и максимальным элементами списка. ЕСЛИ абсолютное значение разности больше 10, ТО вывести на экран отсортированный по возрастанию список, ИНАЧЕ вывести на экран фразу «Разность меньше 10»
3. Напишите программу, которая запрашивает у пользователя список чисел и выводит список, содержащий только четные числа.
4. Напишите программу, которая запрашивает у пользователя список слов и выводит самое длинное слово.
5. Напишите программу, которая запрашивает у пользователя два списка чисел и выводит список, содержащий общие элементы этих списков.

Проект:

"Список дел". Программа позволяет пользователю добавлять, удалять и просматривать список дел.

Этапы выполнения проекта:

Хранение списка дел:
Использовать список для хранения дел.
Добавление дела:
Реализовать функцию, которая запрашивает у пользователя описание дела и добавляет его в список.
Просмотр списка дел:
Реализовать функцию, которая выводит на экран все дела из списка.
Удаление дела:
Реализовать функцию, которая запрашивает у пользователя номер дела для удаления.
Удалить дело из списка по указанному номеру.
Обработать ошибку, если указанный номер не существует.

Контрольные вопросы:

1. Что такое список в Python и чем он отличается от других типов данных, таких как кортежи или множества?
2. Как создать пустой список?
3. Как создать список с начальными элементами?
4. Как получить доступ к элементу списка по его индексу?
5. Что такое отрицательные индексы в списках?
6. Как изменить значение элемента списка по его индексу?
7. Что такое срезы (slices) в списках и как их использовать?
8. Как добавить элемент в конец списка?
9. Как вставить элемент в список по заданному индексу?
10. Как удалить элемент из списка по его значению?
11. Как удалить элемент из списка по его индексу?
12. Как очистить весь список?
13. Как получить длину списка?
14. Как проверить, содержится ли элемент в списке?

VI. ИНСТРУКЦИИ ЦИКЛА В PYTHON

6.1. Цикл While

Цикл — это фундаментальная концепция в программировании, которая позволяет многократно выполнять блок кода. Представьте себе ситуацию, когда вам нужно выполнить одно и то же действие несколько раз, например, вывести числа от 1 до 10 или обработать каждый элемент в списке. Вместо того чтобы писать один и тот же код снова и снова, вы можете использовать цикл.

В Python существует два основных вида циклов: for и while. Каждый из них используется для разных целей и имеет свои особенности.

Для повторения действий, пока выполняется определенное условие, используется конструкция while. Она работает следующим образом:

1. **Проверка условия:** Сначала Python проверяет, верно ли заданное условие (то есть, является ли оно True).
2. **Выход из цикла:** Если условие оказывается неверным (False), выполнение цикла while прекращается, и программа переходит к следующей строке кода.
3. **Выполнение тела цикла:** Если условие верно (True), Python выполняет блок кода, находящийся внутри цикла while.
4. **Повторение:** После выполнения блока кода, Python возвращается к шагу 1 и снова проверяет условие.

Этот процесс повторяется до тех пор, пока условие остается истинным. Каждый раз, когда выполняется блок кода внутри цикла, это называется **итерацией**.

Работу цикла while можно представить в виде блок-схемы (рис 6.1)

Чтобы цикл while не работал бесконечно, нужно, чтобы внутри него что-то менялось. Обычно это достигается изменением значения одной или нескольких переменных. В итоге, эти изменения должны привести к тому, что условие в while станет ложным, и цикл завершится.

Рис 6.1 Блок – схема цикла While

Переменные, которые меняются при каждом повторении цикла и влияют на его завершение, называются **итерационными переменными**. Если в цикле нет такой переменной, которая бы изменялась, то цикл будет работать вечно, то есть станет **бесконечным циклом**.

*Замечание***:** Когда к переменной добавляют 1, это называется **инкрементом**. Когда из переменной вычитают 1, это называется **декрементом**.

Код и результат программы, производящей отсчет от 1 до 5, представлен на рис. 6.2. Для цикла выполнено 5 итераций.

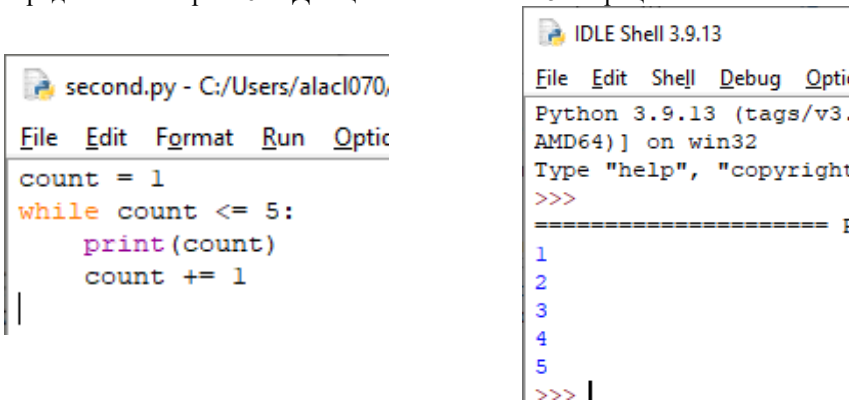


Рис 6.2 Использование цикла while

6.2. Бесконечные циклы и break

Бесконечный цикл не содержит итерационную переменную, которая указывает на количество выполнений в цикле. Он используется в тех случаях, когда только при выполнении очередной итерации становится понятно, надо ли завершить цикл. Для выхода из бесконечного цикла используются операторы break и continue.

Инструкция break используется, когда нужно выйти из бесконечного цикла при наступлении заданного условия:

```
While True:
instruction 1
…
break
…
instruction n
```

Если не добавить breakв тело цикла, то произойдет «*зацикливание*» и цикл будет выполняться бесконечно.

На рис 6.3. представлен цикл, который выводит команду введенную пользователем на экран, до тех пор, пока не будет введено «*exit*». После чего цикл завершается с помощью оператора break.

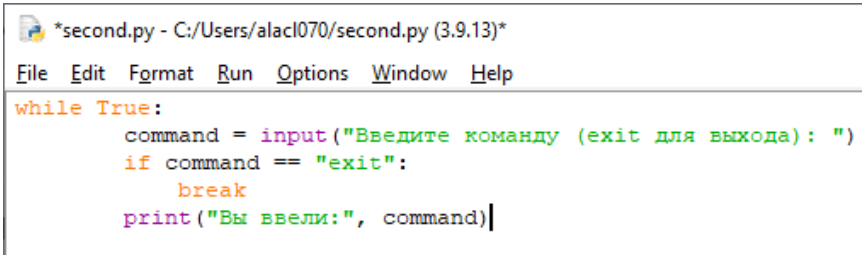


Рис 6.3 Использование оператора break

Инструкция continue пропускает текущую итерацию в цикле и переходит к следующей без завершения тела цикла.

На рисунке 6.4 и 6.5. приведен код программы и результат выполнения, где с помощью цикла производится вывод на экран только положительных чисел из заданного списка. Оператор continue пропускает отрицательные числа.

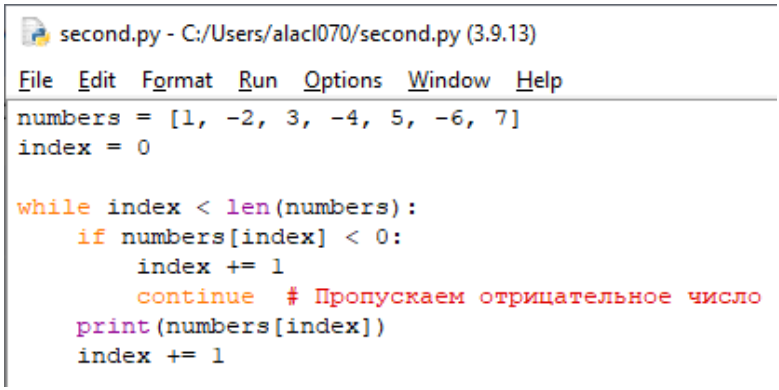


Рис 6.4 Код программы с использование оператора continue

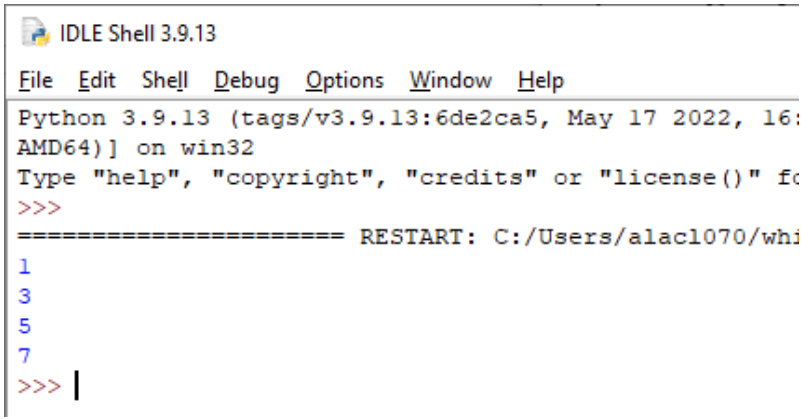


Рис 6.5 Результат использование оператора continue

Практическая работа 6
«Циклические конструкции. Итерационные алгоритмы. Цикл while»

Цель работы: изучить синтаксис циклической конструкции while языка Python для программирования итерационных алгоритмов, понять принцип работы цикла while. Научиться разрабатывать итерационные алгоритмы для решения различных задач.

Задание:

Создайте игру "Угадай число", в которой компьютер загадывает

случайное число, а игрок пытается его отгадать. Программа должна выбрать случайное число в заданном диапазоне (например, от 1 до 100). Игрок вводит число, которое, по его мнению, загадал компьютер. Компьютер сообщает игроку, больше или меньше загаданное число, чем введенное игроком. Если игрок угадывает число, программа поздравляет его и сообщает, сколько попыток ему потребовалось. Программа предлагает игроку сыграть еще раз или выйти.

Проект:

"Генератор паролей". Программа генерирует случайные пароли заданной длины.

Цель проекта:

- Разработать программу, которая генерирует случайные пароли заданной длины.
- Получить практический опыт работы с модулем `random` и строками в Python.
- Развить навыки работы с циклами и функциями.

Этапы выполнения проекта:

1. **Запрос длины пароля:**

   Программа должна запросить у пользователя желаемую длину пароля.

2. **Генерация символов:**

   Определить набор символов, из которых будет состоять пароль (например, буквы, цифры, специальные символы).

   Использовать модуль `random` для случайного выбора символов из этого набора.

3. **Создание пароля:**

   С помощью цикла сгенерировать пароль заданной длины, добавляя случайные символы.

4. **Вывод пароля:**

   Вывести сгенерированный пароль на экран.

Контрольные вопросы:

1. Что такое цикл while и как он работает?
2. Какое условие используется в цикле while?
3. Что произойдет, если условие в цикле while всегда будет истинным?
4. Как прервать выполнение цикла while?
5. В чем отличие цикла while от цикла for?
6. Что такое итерация в контексте цикла while?
7. Что такое итерационная переменная и для чего она нужна?
8. Что произойдет, если в цикле while не изменить итерационную переменную?
9. Как использовать оператор break внутри цикла while?
10. Как использовать оператор continue внутри цикла while?
11. Может ли цикл while не выполниться ни разу?

6.3. Цикл for

Цикл for в Python — это инструмент для перебора элементов в последовательности (например, в списке, кортеже, строке) или в другом итерируемом объекте. Он позволяет выполнить блок кода для каждого элемента последовательности.

Цикл for работает по следующим принципам:

1. Цикл for автоматически перебирает элементы последовательности один за другим.
2. На каждой итерации текущий элемент последовательности присваивается переменной, указанной в заголовке цикла.
3. После присваивания выполняется блок кода, находящийся внутри цикла.
4. После выполнения блока кода цикл переходит к следующему элементу последовательности и повторяет шаги 2 и 3.
5. Цикл завершается, когда все элементы последовательности были обработаны.
6. В общем виде цикл for для перебора всех элементов указанного списка выглядит следующим образом:

```
For <<переменная>> in <<последовательность>>:
        <<тело цикла>>
```

На рисунке 6.5 приведен код программы, где с помощью цикла for производится перебор элементов списка и вывода каждого элемента на экран.

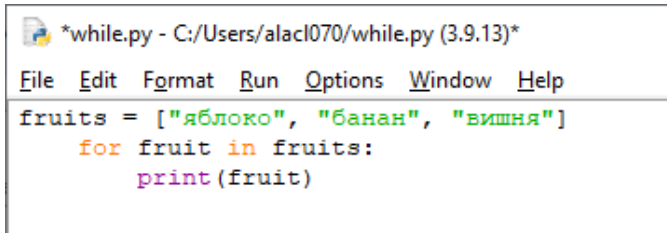


Рис 6.5 Пример применения цикла for для перебора списка

Часто в циклах for для итерации по определенному диапазону чисел используется функция range(). Функция range() в Python — это встроенная функция, которая создает последовательность чисел.

Функция range() может принимать от одного до трех аргументов:

- range(stop): создает последовательность чисел от 0 до stop (не включая stop).
- range(start, stop): создает последовательность чисел от start до stop (не включая stop).
- range(start, stop, step): создает последовательность чисел от start до stop (не включая stop) с шагом step.

На рисунке 6.6 представлен пример, где функция range(1, 10, 2) создает последовательность чисел от 1 до 9 с шагом 2.

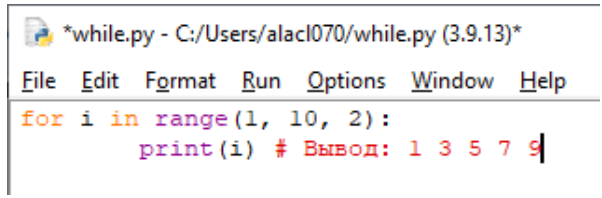


Рис 6.6 Пример применения цикла for для перебора диапазона

Практическая работа 7
«Циклические конструкции. Итерационные алгоритмы. Цикл *for*»

Цель работы: изучить синтаксис циклической конструкции for языка Python для программирования итерационных алгоритмов. Понять, как цикл for перебирает элементы последовательности. Научиться использовать функцию range() для создания последовательностей чисел. Научиться применять операторы break и continue для управления выполнением цикла.

Задание:

1. Напишите программу, которая выводит на экран все элементы заданного списка.
2. Напишите программу, которая вычисляет сумму всех чисел в заданном диапазоне.
3. Напишите программу, которая выводит таблицу умножения для заданного числа.
4. Напишите программу, которая находит и выводит все четные числа из заданного списка.
5. Напишите программу, которая выводит каждый символ заданной строки в отдельной строке.
6. Напишите программу, которая создает и выводит список, содержащий квадраты чисел из заданного диапазона.

Контрольные вопросы:

1. Что такое цикл for и как он работает?
2. Для чего используется цикл for?
3. Как выглядит синтаксис цикла for в Python?
4. Что такое итерируемый объект? Приведите примеры.
5. Как работает функция range() и для чего она используется?
6. Какие аргументы можно передать в функцию range()?
7. Что такое итерация в контексте цикла for?
8. Как прервать выполнение цикла for досрочно?
9. Как пропустить текущую итерацию цикла for?
10. Может ли цикл for не выполниться ни разу?
11. Может ли цикл for содержать внутри себя другие циклы?

VII. СТРОКИ И ОБРАБОТКА ТЕКСТОВОЙ ИНФОРМАЦИИ

7.1. Строки в Python и обработка текстовой информации

Python строка представляет собой неизменяемую последовательность символов. Для доступа к отдельному символу используется оператор [].

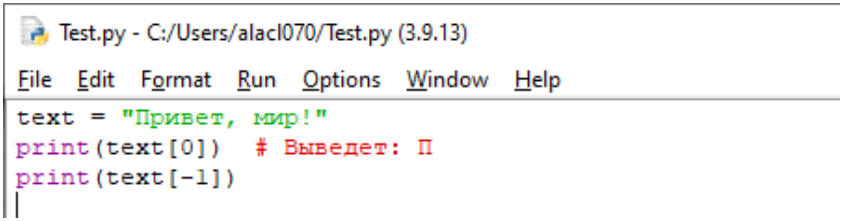


Рис. 7.1 Доступ к одному символу

Число в квадратных скобках называется индексом (index) — оно определяет смещение от начала строки. Первый символ имеет индекс 0, поэтому нумерация начинается с нуля. В качестве индекса можно использовать любое целочисленное выражение, включая переменные и операторы. Также доступны отрицательные индексы: fruit[-1] возвращает последний символ строки, fruit[-2] — предпоследний и так далее.

На рис. 7.2 приводится встроенная функция len(), которая возвращает число символов в строке, включая пробелы и знаки препинания.

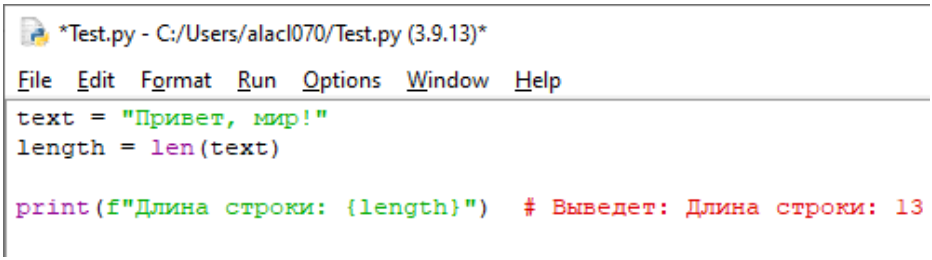


Рис. 7.2 Использование встроенной функции len()

Во многих алгоритмах требуется обработка строки по символам, начиная с её начала. Такой процесс называется обходом. Для обхода строки можно использовать различные циклы. Пример с обходом строки с помощью цикла while и выводом каждого символа на экран приведен на рисунке 7.3.

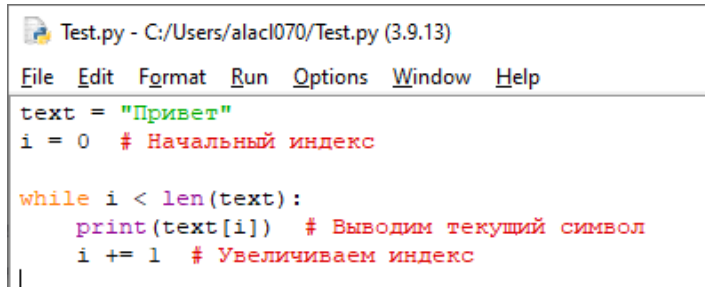


Рис. 7.3 Обход с помощью цикла while

Когда позиция символа не важна, а важно только его значение, то предпочтительнее использовать обход с помощью цикла for (рис. 7.4).

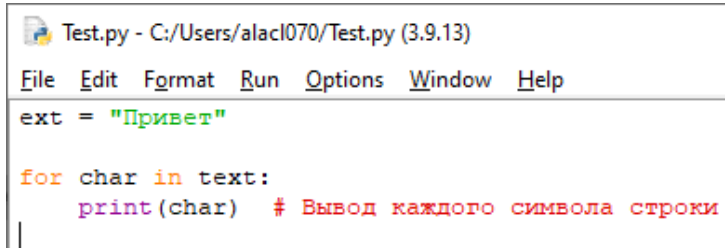


Рис. 7.4 Обход с помощью цикла for

Срезом (slice) называется способ получения части строки с помощью оператора [:]. Срез позволяет извлекать подстроку, указывая начальный и конечный индексы, а также шаг. Оператор [*n*:*m*] возвращает часть строки от *n*-го символа до *m*-го, включая первый, но исключая последний. Если опустить первый индекс, то срез будет начинаться с начала строки. Если опустить второй – срез завершится в конце строки (рис. 7.5).

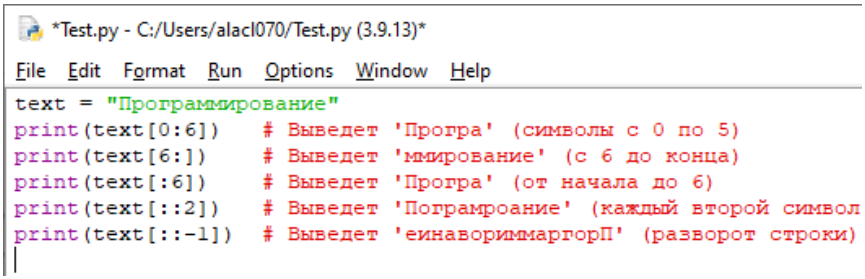


Рис. 7.5 Срез строки

Если начальный индекс больше или равен конечному, результатом среза будет пустая строка. Если оба индекса опущены, возвращается вся строка целиком.

Логический оператор in принимает две строки и возвращает True, если первая строка является подстрокой второй.Пример использования оператора in представлен на рисунке 7.6

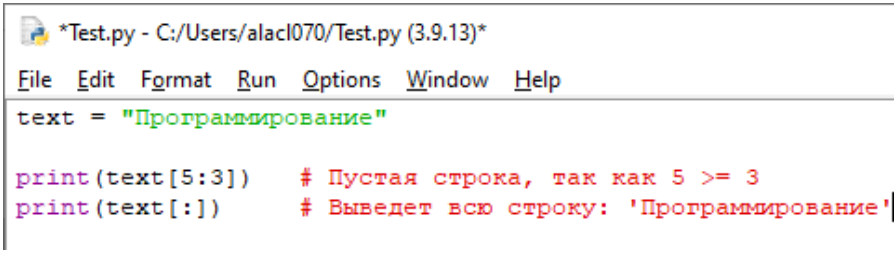


Рис. 7.6 Логический оператор in

В Python строки можно сравнивать. Буквы верхнего регистра имеют более высокий приоритет, чем буквы нижнего, а буквы располагаются перед цифрами. Для проверки равенства строк можно привести их к единому формату, например, преобразовать в нижний регистр.

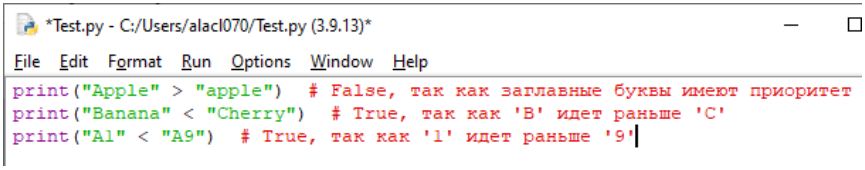


Рис. 7.7 Сравнение строк

7.2. Строковые методы

В Python строка представляет собой объект. Объекты содержат данные (сам текст строки) и методы — встроенные функции, доступные для каждого экземпляра объекта. Для просмотра списка методов, доступных для объекта, можно использовать функцию dir().

Рис. 7.8 Функция dir()

Метод вызывается так же, как функция: он принимает аргументы и возвращает значение. Однако их синтаксис отличается — метод вызывается через точку после имени переменной. В таблице 7.1 представлены синтаксисы различных методов.

Таблица 7.1
Методы обработки строк

Метод	Описание	Пример использования	Результат
upper()	Преобразует строку в верхний регистр	"hello".upper()	'HELLO'
lower()	Преобразует строку в нижний регистр	"HELLO".lower()	'hello'
capitalize()	Делает первую букву заглавной	"python".capitalize()	'Python'
title()	Делает заглавными первые буквы каждого слова	"hello world".title()	'Hello World'

Продолжение таблицы 7.1

Метод	Описание	Пример использования	Результат
strip()	Удаляет пробелы в начале и конце строки	" hello ".strip()	'hello'
lstrip()	Удаляет пробелы в начале строки	" hello".lstrip()	'hello'
rstrip()	Удаляет пробелы в конце строки	"hello ".rstrip()	'hello'
replace(a, b)	Заменяет все вхождения a на b	"apple".replace("p", "b")	'abble'
find(sub)	Ищет подстроку sub, возвращает индекс или -1	"hello".find("e")	1
count(sub)	Подсчитывает количество вхождений подстроки sub	"banana".count("a")	3
startswith(sub)	Проверяет, начинается ли строка с sub	"hello".startswith("he")	True
endswith(sub)	Проверяет, заканчивается ли строка на sub	"hello".endswith("lo")	True
split(sep)	Разбивает строку по разделителю sep	"a,b,c".split(",")	['a', 'b', 'c']
join(iterable)	Объединяет элементы в строку через разделитель	" ".join(["Hello", "World"])	'Hello World'
isdigit()	Проверяет, состоит ли строка только из цифр	"123".isdigit()	True
isalpha()	Проверяет, содержит ли строка только буквы	"hello".isalpha()	True
isalnum()	Проверяет, содержит ли строка только буквы и цифры	"hello123".isalnum()	True

Практическая работа 8

«Строки и обработка текстовой информации»

Цель работы: Ознакомиться с основными операциями со строками в Python. Научиться использовать методы строк для обработки текстовой информации. Освоить работу с индексами, срезами и циклами для обхода строк. Изучить методы поиска, замены и форматирования строк.

Задание:

1. Создайте строковую переменную и выведите её на экран.
2. Определите длину строки с помощью len().
3. Выведите первый и последний символ строки, используя индексы.
4. Выведите подстроку, используя срез с заданными индексами.
5. Разверните строку с помощью среза ([::-1]).
6. Выведите каждый символ строки в отдельной строке, используя for.
7. Используйте while для посимвольного обхода строки.
8. Примените к строке методы: upper(), lower(), title(), strip(), replace().
9. Подсчитайте количество вхождений символа или подстроки (count()).
10. Проверьте, начинается ли строка с определённого символа (startswith()).
11. Проверьте, содержится ли подстрока в строке с помощью in.
12. Сравните две строки с учётом регистра и без него.
13. Разбейте строку на список слов (split()).
14. Объедините список слов в строку (join()).

Проект:"Анализатор текста".

Этапы выполнения проекта:

1. **Чтение файла:**
   Программа должна запросить у пользователя имя файла для анализа.
   Открыть файл в режиме чтения.
   Прочитать содержимое файла и сохранить его в строку.
2. **Обработка текста:**
   Привести текст к нижнему регистру, чтобы слова "Привет" и "привет" считались одинаковыми.
   Удалить знаки препинания и другие небуквенные символы.
   Разделить текст на слова, используя пробелы в качестве разделителя.
3. **Подсчет частоты слов:**
   Создать пустой словарь для хранения частоты слов.
   Для каждого слова в списке:
   Если слово уже есть в словаре, увеличить его счетчик на 1.

Если слова нет в словаре, добавить его и установить счетчик равным 1.

4. **Вывод результатов:**
   Вывести словарь с частотой слов на экран.
   Можно отсортировать слова по частоте встречаемости и вывести топ-N самых часто встречающихся слов.

Контрольные вопросы:

1. Что такое строка в Python?
2. Являются ли строки изменяемыми (mutable) или неизменяемыми (immutable) объектами?
3. Как можно получить доступ к отдельному символу строки?
4. Что такое индексы в строках? Как работают отрицательные индексы?
5. Как узнать длину строки?
6. Что такое срез строки? Какой синтаксис используется для получения подстроки?
7. Как развернуть строку с помощью срезов?
8. Каким образом можно выполнить обход строки с помощью циклов for и while?
9. Какие основные методы строк вы знаете? (назовите хотя бы 5)
10. Как преобразовать строку в верхний и нижний регистр?
11. Как заменить подстроку в строке?
12. Как проверить, содержит ли строка определённую подстроку?
13. Для чего используется оператор in при работе со строками?

VIII. ФУНКЦИИ

В программировании функция — это мини-программа внутри большой программы, которая выполняет конкретную задачу. Представьте, что у вас есть робот-повар. Вы можете дать ему команду "приготовить суп", и он выполнит все необходимые действия: нарежет овощи, сварит бульон и т.д. "Приготовить суп" — это имя функции, а все действия, которые робот выполняет, — это тело функции.

Основными параметрами функции являются:

- имя функции;
- тело функции;
- передаваемые параметры (аргументы);
- возвращаемые параметры (результат).

В Python вы можете создавать собственные функции для решения конкретных задач, а также использовать уже готовые, встроенные функции. Это позволяет делать код более организованным и эффективным.

Правила именования функций такие же, как для переменных, например, нельзя использовать зарезервированные слова в качестве имен функций. Имена функций и переменных не должны совпадать. Первая строка определения функции называется заголовком (header), оставшаяся часть – телом (body) функции. Заголовок заканчивается двоеточием, тело функции имеет отступ. Тело функции может содержать любое количество инструкций. Инструкции внутри функции не получат управления, пока функция не будет вызвана. Для возврата результата функции, используется инструкция return. Вызвать функцию (function call) можно, обратившись к ней по имени.

Приведем пример функции на рис 8.1

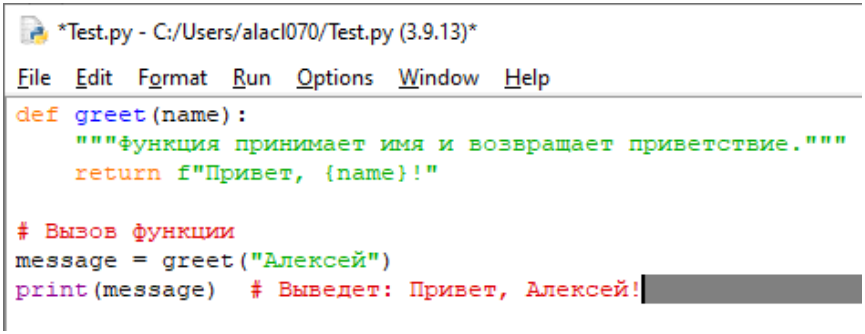


рис 8.1 Пример функции

Здесь:

- def greet(name): — объявление функции greet с параметром name.
- """..""" — строка документации (docstring), поясняющая, что делает функция.
- return f"Привет, {name}!" — функция возвращает строку с приветствием.
- greet("Алексей") — вызов функции с аргументом "Алексей".

Практическая работа 9
«Использование функций»

Цель работы: познакомиться с наиболее востребованными стандартными функциями Python, изучить синтаксис объявления и ис- пользования пользовательских функций.

Задание:

1. Напишите функцию calculate_rectangle_area(width, height), которая принимает ширину и высоту прямоугольника в качестве аргументов и возвращает его площадь.
2. Напишите функцию is_even(number), которая принимает число в качестве аргумента и возвращает True, если число четное, и False, если нечетное.
3. Напишите функцию find_max(a, b, c), которая принимает три числа в качестве аргументов и возвращает максимальное из них.
4. Напишите функцию celsius_to_fahrenheit(celsius), которая принимает температуру в градусах Цельсия и возвращает ее значение в градусах Фаренгейта.
5. Напишите функцию sum_list(numbers), которая принимает список чисел в качестве аргумента и возвращает сумму его элементов.
6. Напишите функцию filter_even(numbers), которая принимает список чисел и возвращает новый список, содержащий только четные числа.

Проект:

"Телефонный справочник". Программа хранит информацию о контактах и позволяет пользователю добавлять, искать и удалять контакты.

Этапы выполнения проекта:

1. **Хранение данных:**

Использовать словарь для хранения контактов, где ключом может быть имя, а значением - словарь с информацией о контакте (номер телефона, адрес и т. д.).

2. **Добавление контакта:**

   Реализовать функцию, которая запрашивает у пользователя имя и номер телефона (и, возможно, другие данные).
   Добавить новый контакт в словарь.

3. **Поиск контакта:**

   Реализовать функцию, которая запрашивает у пользователя имя для поиска.
   Найти контакт в словаре и вывести его информацию.
   Если контакт не найден, вывести соответствующее сообщение.

4. **Удаление контакта:**

   Реализовать функцию, которая запрашивает у пользователя имя для удаления.
   Удалить контакт из словаря, если он существует.
   Если контакт не найден, вывести соответствующее сообщение.

5. **Интерфейс пользователя:**

   Создать цикл, который выводит меню с опциями (добавить, найти, удалить, выйти).
   Обрабатывать выбор пользователя и вызывать соответствующие функции.

Контрольные вопросы:

1. Что такое функция в программировании и для чего она используется?
2. Чем отличаются встроенные функции от пользовательских?
3. Как определить пользовательскую функцию в Python?
4. Что такое аргументы (параметры) функции?
5. Как передать аргументы в функцию?
6. Что такое возвращаемое значение функции и как его получить?
7. Как использовать инструкцию return в функции?
8. Что такое область видимости переменных (локальные и глобальные переменные)?

IX. СЛОВАРИ, КОРТЕЖИ И МНОЖЕСТВА

9.1. Работа со словарями

Для работы с наборами данных Python предоставляет такие встроенные типы как словари, кортежи и множества.

Словарь (*dictionary*) похож на список, но имеет более широкие возможности. В списке индекс имеет целочисленное значение, в словаре может быть любого типа. dict() создает словарь без записей (рис. 9.1). Ключ 'one' отображается в значение 'uno'.

```
>>> eng2sp = dict()
>>> print(eng2sp)
{}
>>> eng2sp['one'] = 'uno'
>>> print(eng2sp)
{'one': 'uno'}
```

Рис. 9.1 Создание словаря

Для добавления записей в словарь, можно использовать квадратные скобки.

Вывод словаря на разных компьютерах может дать разный результат, так как порядок пар ключ-значение не всегда совпадает. Функция len() возвращает число пар ключ-значение. Оператор in сообщает о похожих ключах. Метод keys() возвращает множество ключей словаря в виде списка, а метод values() возвращает в виде списка множество значений словаря (рис. 9.2).

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06)
tel)] on win32
Type "copyright", "credits" or "license()" for more inform
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> eng2sp.keys()
dict_keys(['two', 'three', 'one'])
>>> eng2sp.values()
dict_values(['dos', 'tres', 'uno'])
```

Рис. 9.2 Метод keys()

Для обхода словаря можно использовать цикл for(рис. 9.3).

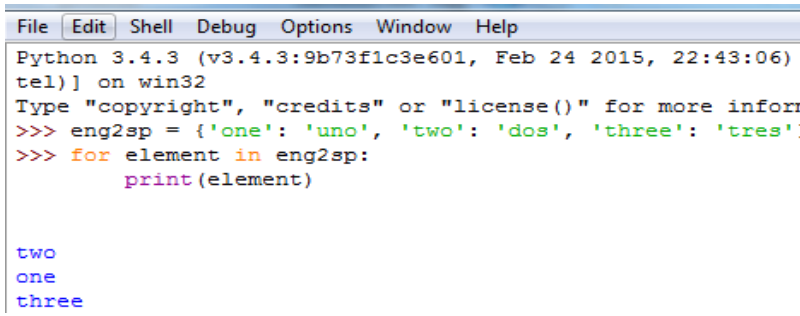


Рис. 9.3 Обход словаря с помощью цикла for

9.2. Работа с кортежами

Кортеж (tuple) — это неизменяемый тип данных в Python, предназначенный для хранения упорядоченной последовательности элементов. Он обладает тремя ключевыми характеристиками:

1. **Неизменяемость** – после создания кортеж нельзя изменять: добавлять, удалять или заменять его элементы. Попытка изменить кортеж приведёт к ошибке TypeError.
2. **Упорядоченность** – элементы кортежа сохраняют свой порядок, и доступ к ним осуществляется по индексам.
3. **Разнотипность элементов** – кортеж может содержать данные различных типов, включая числа, строки, списки и другие кортежи. Также допускается вложенность коллекций любой глубины, например, кортеж может содержать список, который, в свою очередь, включает другой список и так далее.

Пример работы с кортежем приведен на рис 9.4

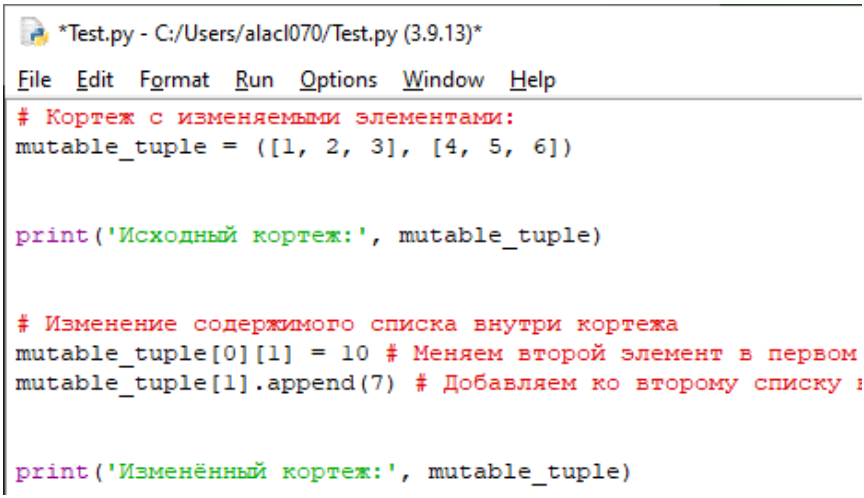


Рис 9.4 Использование кортежей

9.3. Множества в Python

В Python **множество** (set) — это структура данных, аналогичная множествам в математике. Оно может содержать различные элементы, но их порядок не определён.

Основные свойства множеств:

- **Добавление и удаление элементов** – можно динамически изменять состав множества.
- **Перебор элементов** – возможно итерирование по множеству.
- **Операции над множествами** – поддерживаются объединение, пересечение, разность и другие математические операции.
- **Быстрая проверка принадлежности** – поиск элемента выполняется быстрее, чем при переборе списка, благодаря особенностям внутреннего хранения данных.

Элементами множества могут быть только неизменяемые типы данных, такие как числа, строки и кортежи. Изменяемые структуры, например списки и другие множества, не могут входить в состав множества, так как это связано со способом хранения данных в памяти.

Множество задается перечислением всех его элементов в фигурных скобках. Исключением является пустое множество, которое можно создать при помощи функции set(). Если функции set() передать в качестве параметра список, строку или кортеж, то она вернёт множество, составленное из элементов списка, строки, кортежа. Пример создания множества приведен на рисунке 9.5

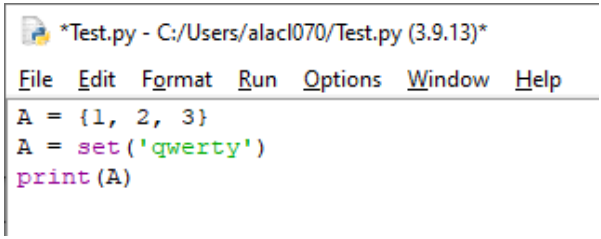


Рис 9.5 Создание множества из строки

Функция range позволяет создавать множества из диапазона:

```
>>> set(range(10))
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
>>>
```

Операции с множествами в Python приведены в таблице 9.1.

Таблица 9.1

Операции с множествами в Python

Операция	Описание	Пример	Результат
add(x)	Добавляет элемент x в множество	s = {1, 2, 3}; s.add(4)	{1, 2, 3, 4}
remove(x)	Удаляет элемент x, вызывает ошибку, если его нет	s = {1, 2, 3}; s.remove(2)	{1, 3}
discard(x)	Удаляет элемент x, не вызывает ошибку, если его нет	s = {1, 2, 3}; s.discard(4)	{1, 2, 3}
pop()	Удаляет случайный элемент и возвращает его	s = {1, 2, 3}; s.pop()	1 (или другой элемент)
clear()	Очищает множество	s = {1, 2, 3}; s.clear()	set()
union(set2) или `	Объединение множеств	a = {1, 2, 3} b = {3, 4, 5} union = a.union(b)	{1, 2, 3, 4, 5}
intersection(set2) или &	Пересечение множеств	{1, 2, 3} & {2, 3, 4}	{2, 3}
difference(set2) или -	Разность множеств	{1, 2, 3} - {2, 3, 4}	{1}
symmetric_difference(set2) или ^	Симметричная разность (элементы, которые есть только в одном из множеств)	{1, 2, 3} ^ {2, 3, 4}	{1, 4}
issubset(set2) или <=	Проверяет, является ли множество подмножеством	{1, 2} <= {1, 2, 3}	True
issuperset(set2) или >=	Проверяет, является ли множество надмножеством	{1, 2, 3} >= {1, 2}	True
isdisjoint(set2)	Проверяет, не пересекаются ли множества	{1, 2} .isdisjoint({3, 4})	True

Практическая работа 10
«Кортежи, словари и множества»

Цель работы: приобрести навыки работы со стандартными встроенными высокоуровневыми типами данных в Python и закрепить их на примере разработки интерактивных приложений.

Задание:

Разработать интерактивную программу, которая будет модели- ровать игру «Виселица» («Hangman»). Компьютер загадывает слово и выводит на консоль количество подчеркиваний, равное числу букв в загаданном слове. Пользователь начинает вводить буквы, чтобы отгадать слово. Если буква есть в слове, компьютер вписывает ее на свое место в слово, а если нет, компьютер рисует в консоли один элемент импровизированной виселицы (к примеру: стойка, перекладина, веревка, голова, туловище, две руки, две ноги). Дополнительно выводятся буквы, которые уже вводились.

Если игрок не успел угадать слово раньше, чем компьютер нарисовал полностью виселицу, то игрок считается проигравшим. Если игрок успевает угадать слово, то он выигрывает.

Дополнительное задание

Модифицировать программу «Виселица» таким образом, чтобы в ней были уровни сложности и (или) подсказки.

Контрольные вопросы:

1. Что такое кортеж в Python?
2. Чем кортеж отличается от списка?
3. Можно ли изменить элементы кортежа после его создания?
4. Как создать кортеж? Приведите примеры.
5. Как обратиться к элементу кортежа по индексу?
6. Можно ли создать кортеж из одного элемента? Как?
7. Какие операции можно выполнять с кортежами?
8. Можно ли использовать кортежи в качестве ключей словаря?
9. Опишите, что такое множество в Python и чем оно отличается от списка и кортежа.
10. Приведите примеры создания множеств. Что произойдет, если вы попытаетесь создать множество с дублирующимися элементами?
11. Как можно объединить два множества?
12. Приведите пример, как получить элементы, которые есть в одном множестве, но отсутствуют в другом.
13. Можно ли изменить элементы внутри множества?

ЗАКЛЮЧЕНИЕ

В современном мире информационных технологий знание языков программирования становится неотъемлемой частью образования. Python, благодаря своей простоте и универсальности, является отличным выбором для первого знакомства студентов с миром программирования.

Данное учебно-методическое пособие разработано с целью помочь преподавателям организовать и провести эффективные факультативные занятия по Python для студентов 1 курса колледжа. В пособии приведен материал, который будет не только информативным, но и интересным, способствующим развитию у студентов практических навыков и мотивации к дальнейшему изучению программирования.

Использование современных образовательных технологий, практико-ориентированный подход и акцент на проектной деятельности делают это пособие актуальным и востребованным. Оно поможет студентам не только освоить основы Python, но и развить критическое мышление, умение решать задачи и работать в команде.

Полученные знания и навыки станут прочной основой для дальнейшего обучения и профессионального развития студентов в области информационных технологий.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Доусон, М. Программируем на Python / М. Доусон – СПб.: Питер, 2014. – 416 с.
2. Лутц, М. Изучаем Python / М. Лутц. – 4-е издание.; пер. с англ. –СПб.: Символ-Плюс, 2011. – 1280 с.
3. Лутц, М. Программирование на Python / М. Лутц. – 4-е изда- ние.; пер. с англ. – СПб.: Символ-Плюс, 2011. – Т. 1. – 992 с.
4. Прохоренок, Н. А. Python 3 и PyQt. Разработка приложений /Н. А. Прохоренок. – СПб.: БХВ-Петербург, 2012. – 704 с.
5. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учеб. пособие для прикладного бакалавриата / Д. Ю. Федоров. — 2-е изд., перераб. и доп. — М. : Издательство Юрайт, 2019. — 161 с.
6. Хахаев, И. А. Практикум по алгоритмизации и программиро- ванию на Python / И. А. Хахаев – М.: Альт Линукс, 2010. – 126 с.
7. Шапошникова, С. Основы программирования на Python [Электронный ресурс]
8. Северанс, Ч. Р. Введение в язык программирования Python [Электронный ресурс]