МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И
МАССОВЫХ КОММУНИКАЦИЙ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра информатики и вычислительной техники

Т.А. Коваленко, А.Г. Солодов

«ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА И ЯЗЫКИ ПРОГРАММИРОВАНИЯ»

Часть 1 - Python Edition

Учебное пособие

Самара
2021

УДДК - 004.772
ББК 3.32.97
К-56

Рекомендовано к изданию методическим советом ПГУТИ, протокол

№_____ от _ __ _____ 2021

Рецензент:

к.т.н. нальник отдела сбора информации ГНКЦ ЦСКБ «ПРОГРЕСС»

Халилов Р.Р.

Коваленко Т.А

К-56 Вычислительная техника и языки программирования. Часть 1 (Python Edition): учебное пособие/ Т.А. Коваленко, А.Г. Солодов– Самара: ИНУЛ ПГУТИ, 2021.

Учебное пособие предназначено для студентов первого курса дневной и заочной формы обучения. В нем рассмотрены вопросы программирования на языке Python в рамках учебной дисциплины «Информатика».

Пособие представлено в двух частях теоретической и практической. В теоретическое части дается представление о языке Python, его возможностях, которые позволяют решать задачи, используя язык программирования. Вторая часть состоит из 14 лабораторных работ. Все задачи классифицированы, т.е. объединены в некоторые группы.

Пособие позволяет рассмотреть не только теоретические вопросы, но и выполнить самостоятельно лабораторные работы.

Использование данного учебного пособия является хорошим подспорьем для студентов технических специальностей.

Данное пособие поможет студентам использовать навыки программирования в технических приложениях (ОПК-4), повысить знания принципов алгоритмизации и программирования (ОПК-4) и овладеть основными методами работы на компьютере с использованием универсальных прикладных программ (ОПК-4).

Материал, представленный в учебном пособии, является актуальным. Он изложен доступным для студентов языком.

Учебное пособие является необходимым и полезным в учебном процессе.

СОДЕРЖАНИЕ

Введение

Данное пособие рассчитано для использования студентами 1 курса на лабораторно–практических занятиях по предмету «Вычислительная техника и языки программирования».

Этот предмет состоит из двух частей. В первой части мы изучаем язык программирования Python, и это пособие предназначено для ознакомления с основными методами, применяемыми для решения различных задач.

Сегодня можно выделить три основных языка программирования, которые являются ключевыми: язык Python, Java, C++.

Проанализировав рынок использования языков программирования в настоящее время, был сделан вывод, что язык Python востребован на рынке, поэтому учебное пособие посвящено изучению этого языка. Python является одним из самых популярных языков программирования благодаря своей простоте, читаемости кода и широкому спектру применений - от веб-разработки до машинного обучения и анализа данных.

Язык программирования Python был разработан Гвидо ван Россумом в 1991 году как эффективное, надежное и простое в использовании средство. В настоящее время Python является интерпретируемым языком высокого уровня, который позволяет быстро создавать надежное программное обеспечение для различных платформ - от веб-серверов до мобильных приложений и систем искусственного интеллекта.

Первым рассматривается консольное приложение, здесь в лабораторном комплексе особое внимание уделяется блок-схемам, умению строить решение задачи по блок-схеме. Затем рассматривается работа с графическими интерфейсами с использованием библиотек tkinter, PyQt или других. Пособие предусматривает получение студентами навыков самостоятельного написания программ.

Каждая тема содержит теоретический материал и примеры использования языка программирования для решения конкретных задач. В учебном пособии дается описание основных вычислительных алгоритмов, тексты программ и описание стандартных функций языка Python, реализующих изученные вычислительные алгоритмы.

Для закрепления теоретического материала предусматривается выполнение лабораторных работ по основным рассматриваемым темам.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

В теоретической части рассматриваются вопросы написания программ для решения задач с помощью языка Python. Описываются простейшие конструкции программы, структура языка. Дается определение алгоритма, переменной, понятие блок-схемы. Подробно рассматривается вопрос построения программы с помощью Python.

Цель теоретической части: Дать представление о языке Python, его структуре и особенностях. Ознакомиться с основными функциями и возможностями этого языка. Уяснить предназначение функций. Понять что такое идентификатор, переменная, константа, типы данных, код программы. Научиться писать программы на языке Python.

Лекция 1

Процесс создания программного обеспечения

Цель лекции: ознакомиться с основными понятиями, встречающимися при изучении программирования. Уяснить что такое машинный язык, из чего состоит процесс создания программного обеспечения, что в себя включают программные продукты и какова их классификация. Особое внимание уделено особенностям разработки на языке Python.

Прежде чем программировать компьютер, мы должны понять, как он работает.

Программирование – это постоянная борьба с машиной. Нужно заставлять её делать то, что тебе нужно. Поэтому любой программист просто обязан знать его внутренности. Компьютер состоит из следующих основных компонентов: процессор, память, видеокарта, винчестер (жёсткий диск) и различные разъёмы для подключения дополнительных устройств. Все эти компоненты связаны между собой с помощью шлейфов и шин.

Вся информация в компьютере хранится на винчестере. Когда ты запускаешь какую-нибудь программу, то она сначала загружается в память и только потом процессор начинает выполнять содержащиеся в ней инструкции. Чем больше программа, тем дольше она загружается.

Результат работы программы выводится на экран через видеокарту. На любой видеокарте есть чип памяти, в котором отображается всё содержимое экрана. Когда нужно вывести что-то на

экран, просто копируются данные в видеопамять, и видеокарта автоматически выводит его содержимое на монитор.

Это всё, что необходимо знать о работе компьютера.

1.1 Машинный язык.

Данные на диске хранятся в двоичном виде. Даже текстовые файлы на диске выглядят в виде нулей и единиц. Точно так же выглядит и любая программа, только её называют машинным кодом. Давай с ним познакомимся немного поближе.

Любая программа представляет собой последовательность команд. Эти команды называются процессорными инструкциями.

По этим инструкциям процессор определяет, что и как ему нужно делать. Когда ты запускаешь программу, компьютер загружает её машинный код в память и начинает выполнять. Наша задача, написать эти инструкции, чтобы компьютер понял, что мы от него хотим.

Реальная программа, которую выполняет компьютер, представляет собой последовательность единиц и нулей. Такую последовательность называют машинным языком. Но человек не способен эффективно думать единицами и нулями. Для нас легче воспринимается осмысленный текст, а не сумасшедшие числа в двоичной системе измерения, с которой мы не привыкли работать.

Например, команда складывания двух регистров выглядит так: #03C3. Нам это мало о чём говорит, и запомнить такую команду очень тяжело. На много проще написать «сложить число1 + число2».

Первое время программисты писали в машинных кодах, пока кому-то не пришла в голову идея: «Почему бы не писать текст программы на понятном языке, а потом заставлять компьютер

переводить этот текст в машинный код?». Идея действительно заслуживала внимания. Так появился первый компилятор – программа, которая переводила текст программ в машинный код.

Множество управляющих сигналов можно связать с набором 0 и 1, которые можно интерпретировать, как число. Например, 0110001100110101.

Программа, с которой работает процессор, это последовательность чисел, называемая машинным кодом.

1.2 Процесс создания программного обеспечения

Программа – упорядоченная последовательность команд компьютера для решения задачи.

Программное обеспечение (ПО) – совокупность программ обработки данных и необходимых для их эксплуатации документов.

Процесс создания программ можно представить как последовательность следующих действий:

- постановка задачи,
- алгоритмизация решения задачи
- программирование.

Постановка задачи связана с конкретизацией основных параметров ее реализации, определением источников и структуры входной (исходные данные) и выходной (вид документов, сигналы управления) информации.

Алгоритм – система точно сформулированных правил, определяющая процесс преобразования входной информации в выходную информацию за конечное число шагов.

В алгоритме отражаются логика и способ формирования результатов решения с указанием необходимых расчетных формул. В него входят логические условия и соотношения для контроля достоверности выходных результатов.

Алгоритм решения задачи имеет ряд обязательных свойств:

- дискретность – разбиение процесса обработки информации на простые этапы (шаги), выполнение которых компьютером или человеком не вызывает затруднений;
- определенность – однозначность выполнения каждого шага преобразования информации;
- выполнимость – конечность действий алгоритма решения задачи, позволяющая получить желаемый результат за конечное число шагов;
- массовость – пригодность алгоритма для решения определенного класса задач.

Способы описания алгоритмов:

- Словесный - описание порядка действий на естественном языке;
- Графический - с использованием блок-схем алгоритма в виде графических символов. Размеры блоков стандартизированы;
- Программный - текст на языке программирования. Лаконичный, наглядный.

Алгоритм, написанный на языке программирования, называется ***программой***.

При работе с алгоритмами используют понятие оператора.

Оператор – это формальная запись инструкций по выполнению некоторой последовательности действий.

Программирование – теоретическая и практическая деятельность, связанная с созданием программ.

Все программы можно разделить на два класса:

- утилитарные программы
- программные продукты.

Утилитарные программы («программы для себя») предназначены для удовлетворения нужд разработчиков. Чаще всего они играют роль сервиса в технологии обработки данных.

Программные продукты (изделия) предназначены для удовлетворения потребностей пользователей и представляют собой комплекс взаимосвязанных программ для решения определенной задачи массового спроса.

Как правило, программные продукты требуют сопровождения, которое осуществляется фирмами-распространителями программ (дистрибьюторами), реже – фирмами-разработчиками.

Сопровождение программного продукта – поддержка его работоспособности, переход на его новые версии, внесение изменений, исправление обнаруженных ошибок и т.п.

1.3 Классификация программных продуктов

По области использования можно выделить три класса программных продуктов:

- системное программное обеспечение
- пакеты прикладных программ
- инструментарий технологии программирования.

Системное программное обеспечение – совокупность программ и программных комплексов для обеспечения работы компьютера и сетей ЭВМ. Данный класс программных продуктов тесно связан с типом компьютера и является его неотъемлемой частью.(рис.1.1)

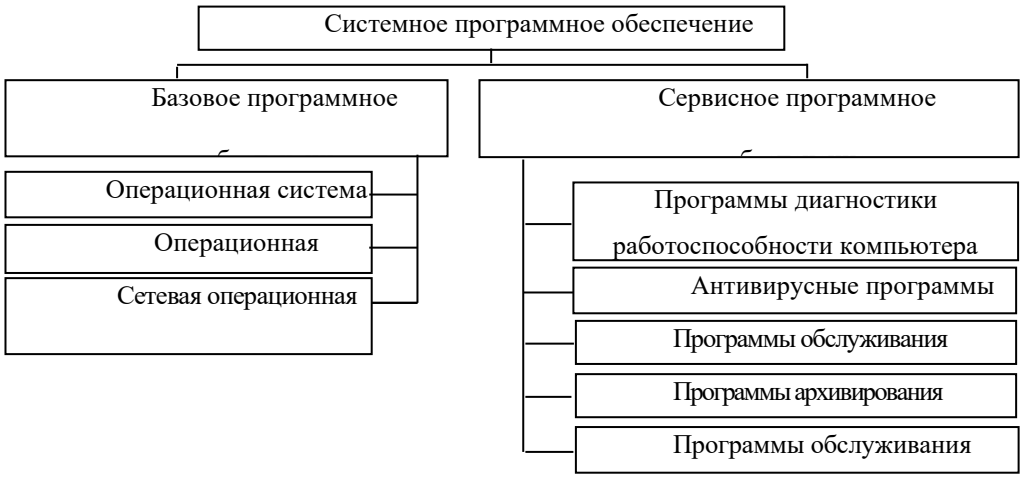

Рис. 1.1 – Классификация системного программного обеспечения

Базовое программное обеспечение, как правило, поставляется вместе с компьютером, а сервисное может быть приобретено дополнительно.

Пакеты прикладных программ (ППП) – комплекс взаимосвязанных программ для решения задач определенного класса конкретной предметной области.

Предметная (прикладная) область – совокупность связанных между собой функций (задач управления), с помощью которых достигается выполнение поставленных целей.

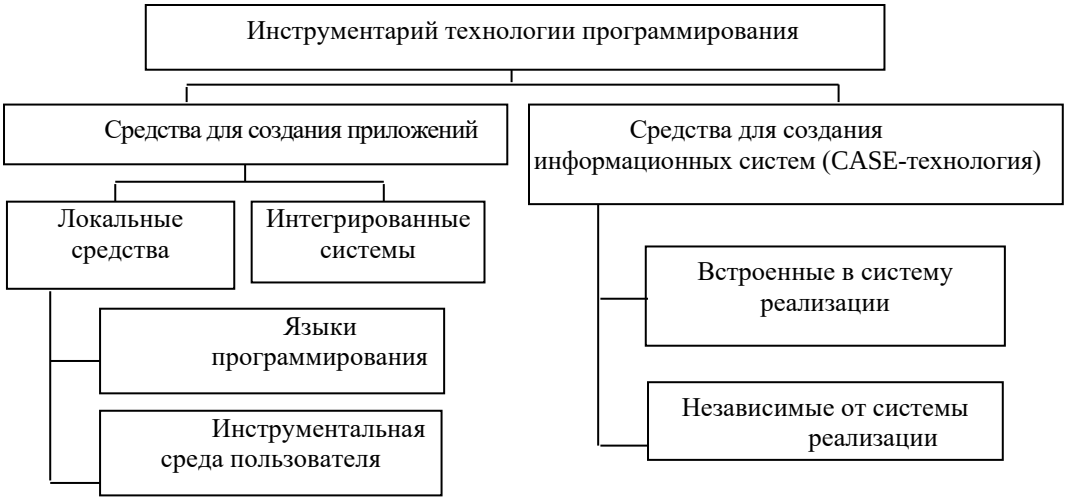

Рис.1.2 – Классификация инструментария технологии программирования

Инструментарий технологии программирования – программные продукты поддержки (обеспечения) технологии программирования (рис.1.2).

Средства для создания приложений – совокупность языков и систем программирования, а также различные программные комплексы для отладки и поддержки создаваемых программ.

Язык программирования – формальный язык для описания алгоритма решения задачи на компьютере. Языки программирования по синтаксису образования их конструкций можно условно разделить на следующие классы:

- машинные (машинные коды), воспринимаемые аппаратной частью ЭВМ;
- машинно-ориентированные (ассемблеры), отражающие структуру конкретного типа процессора;
- алгоритмические (Python, Паскаль, Си и др.), отражающие структуры алгоритма и не зависящие от типа процессора;

- проблемно-ориентированные (Лисп, Симула и др.), предназначенные для решения задач определенного класса.

Для создания программы на выбранном языке программирования нужно иметь следующие компоненты:

Интерпретатор Python – программа, которая выполняет исходный код Python построчно, преобразуя его в байт-код и выполняя. Python является интерпретируемым языком, что означает, что код выполняется напрямую без предварительной компиляции в машинный код.

В отличие от компилируемых языков, Python не требует отдельного этапа компиляции. Интерпретатор Python анализирует структуру каждого оператора языка и затем сразу его исполняет. Это делает процесс разработки более быстрым, но может снизить производительность выполнения по сравнению с скомпилированными программами.

Современные версии Python используют компиляцию в байт-код для повышения производительности. Байт-код сохраняется в файлах .рус и может быть переиспользован при повторном запуске программы.

Инструментальная среда пользователя представлена специальными средствами: библиотека функций, модули и многое другое.

Интегрированные системы программирования для Python предназначены для повышения производительности труда программистов. Они включают в себя:

- текстовый редактор с подсветкой синтаксиса Python, где ключевые слова и идентификаторы выделяются разными цветами и шрифтами и, кроме того, автоматически проверяется правильность синтаксиса программы непосредственно во время ее ввода;

- интерпретатор Python, отладчик и библиотеки функций;
- отладчик, который позволяет анализировать работу программы во время ее выполнения с целью обнаружения и устранения ошибок. С его помощью можно последовательно выполнять отдельные операторы исходного текста по шагам, наблюдая при этом, как меняются значения различных переменных.

Популярные IDE для Python:

- PyCharm (JetBrains)
- Visual Studio Code с расширением Python
- Jupyter Notebook
- Spyder
- IDLE (встроенная в Python)

В интегрированных системах все этапы создания программы автоматизированы: после того как исходный текст введен, его выполнение и отладка выполняются одним нажатием клавиши.

1.4 Классификация языков программирования

ЭВМ исполняет программу в машинных кодах. А составляют программу люди на удобном для себя языке.

Различают языки: высокого уровня и низкого уровня (машинно-ориентированные).

Языки высокого уровня бывают:

- процедурно-ориентированные. Содержат набор универсальных команд;
- проблемно-ориентированные. Имеют команды узкого назначения;

- объектно-ориентированные. Программирование на уровне объектов;
- событийно-ориентированные. Программирование на уровне событий;
- функциональные. Программирование на основе функций;
- комплексные. Поддерживают многие из перечисленных свойств.

Python является мультипарадигмальным языком, поддерживающим:

- Объектно-ориентированное программирование
- Процедурное программирование
- Функциональное программирование
- Аспектно-ориентированное программирование

Различают пять поколений языков программирования:

Начало 1950-х годов. Язык Ассемблера. Его принцип "Одна инструкция – одна строка". Инструкция на языке однозначно соответствует машинному коду команды.

Начало 1950-х – конец 1960-х годов. Язык символического Ассемблера. В нем появилось понятие переменной.

1960-е годы. Универсальные языки программирования.

С начала 1970-х годов до настоящего времени. Проблемно-ориентированные языки для создания проектов в узкой предметной области.

С середины 1990-х годов до настоящего времени. Языки с автоматизацией программирования. Примеры – языки визуального программирования, скриптовые языки (Python, JavaScript).

Таблица 1.1

Языки низкого уровня

Язык	Расшифровка
Assembler	Ассемблер
Macro Assembler	Макро Ассемблер

Таблица 1.2

Языки высокого уровня

Язык	Расшифровка	Примечание
Fortran	Formula Translator	От слов - транслятор формул.
BASIC	Beginner's All-purpose Symbolic Instruction Code	Многоцелевой мнемокод для начинающих. Создан в 1960-е годы.
Visual Basic	Визуальный BASIC	Язык 5-го поколения. Его версия – рабочий язык пакета Microsoft Office.
Cobol	Common Business Oriented Language	Язык для задач в экономике бизнесе.
Algol	Algorithmic Language	Создан для описания алгоритмов. Не получил широкого распространения.
Pascal		Универсальный язык. Создан в 1970-х годах. Один из часто применяемых.
C		Язык для системного программирования. Создан в 1970-х годах компанией Bell.
C++		Объектно-ориентированное расширение C++. Создан в 1980 году Страуструпом.
C#	С шарп	Многоплатформенная версия C++
Java		Модификация C для Internet. В нем удалены низкоуровневые возможности языка C.
Python	Высокоуровневый интерпретируемый язык	Многопарадигменный (ООП/процедурный/функц.), читабельный синтаксис, богатая стандартная библиотека; применяется в науке о данных, вебе, автоматизации, ML.

Показатели качества ПО

Перечень показателей качества:

- документированность,
- эффективность,
- простота использования
- удобство эксплуатации,
- мобильность,
- совместимость,
- испытуемость,
- стоимость.

Показатели качества производителя ПО гарантируются международным сертификатом стандарта качества ISO 9000.

Сертификат выдается либо международный, либо на определенную территорию (например, на Восточную Европу).

Разработанная в США методология CMM (CMM = Capability Maturity Model – Модель зрелости.) сертифицирует производителя ПО по 5 уровням.

На основе CMM создана методология PSP, позволяющая резко повысить мастерство программистов. Она включает:

- составление календарного плана работы,
- хронометраж работ,
- рекомендации по недопущению ошибок,
- анализ проекта после завершение работы.

Программист должен не только знать язык программирования, но и уметь планировать свой труд.

Стиль программирования

Хороший стиль программирования включает в себя:

- стандартизацию средств

- верные комментарии,
- использование пробелов,
- многоярусную запись текста,
- выбор наглядных идентификаторов,
- упорядочивание списков,
- в строке 1 оператор,
- вынос из циклов постоянных операторов,
- бригадное программирование.

Принцип простоты (KISS)

- Keep делай
- It это
- Simple проще,
- Stupid глупец,

1.5 Маркетинг ПО

Различают три вида ПО:

1. Коммерческое

- Издатель заказывает ПО у исполнителя и выделяет средства на работу.
- Издатель получает все имущественные права на созданный продукт.
- Исполнитель может получить некоторый процент (роялти) с каждой проданной копии ПО.
- Издатель оплачивает расходы на упаковку, рекламу, подготовку документации и др.
- За исполнителем сохраняются авторские права на ПО.

2. Условно-бесплатное (Shareware)

- Автор бесплатно представляет клиенту ознакомительную версию ПО.
- Клиент может за определенную плату приобрести полную рабочую версию.

3. Бесплатное ПО (Freeware, Public Domain)

- Автор может попросить заплатить ему некоторую сумму, не настаивая на этом.
- Клиент может с этим не согласиться.

Python и его экосистема

Python является открытым языком программирования с лицензией Python Software Foundation License, совместимой с GPL. Это означает, что Python можно свободно использовать, модифицировать и распространять.

Экосистема Python включает:

- Стандартную библиотеку с множеством модулей
- PyPI (Python Package Index) - репозиторий пакетов
- Виртуальные окружения для изоляции проектов
- Менеджеры пакетов (pip, conda)
- Фреймворки для различных областей (Django, Flask, FastAPI для веб-разработки; NumPy, Pandas для анализа данных; TensorFlow, PyTorch для машинного обучения)

Краткие итоги

В лекции были рассмотрены основные понятия, встречающиеся в курсе программирования. Даны определения, что такое алгоритм, машинный язык, программирование. Освещены вопросы инструментария технологий программирования. Дано описания языков

высокого и низкого уровня. Особое внимание уделено особенностям языка Python и его экосистемы.

Контрольные вопросы

1. Дайте определение понятиям «программирование», «машинный язык», «программа», «алгоритм».
2. Какие команды называют процессорными инструкциями?
3. Из чего состоит процесс программного обеспечения?
4. Что такое системное программное обеспечение?
5. Чем характеризуются пакеты прикладных программ?
6. Дайте определение инструментарию технологий программирования.
7. Для чего предназначены интегрированные системы программирования?
8. Каков перечень показателей качества?
9. Что в себя включает стиль программирования?
10. Перечислите и охарактеризуйте виды ПО.
11. Перечислите языки низкого уровня.
12. Перечислите языки высокого уровня.
13. В чем особенности Python как языка программирования?
14. Какие IDE рекомендуются для разработки на Python?
15. Что такое PyPI и для чего он используется?

Лекция 2
Типы данных в Python

Цель лекции: ознакомиться с типами данных применяемых в программах на языке Python. Уяснить, как применяются типы данных и для чего они нужны. Разобрать вопрос задания типов данных.

Python является динамически типизированным языком. Каждая переменная имеет тип, который определяется автоматически при присваивании значения. В отличие от статически типизированных языков, в Python тип переменной может изменяться во время выполнения программы.

Тип данных определяет множество значений, которые могут принимать элементы программы. Введение типов дает возможность автоматического поиска ошибок во время выполнения, что приводит к созданию надежных программ.

Тип определяет для элемента программы:

- объем памяти для размещения. Python автоматически управляет памятью
- место для хранения переменной типа
- минимальное и максимальное значения, которые могут принимать данные
- разрешенные операции

Типы в Python могут быть:

- встроенные (не требуют импорта)
- пользовательские (классы, определенные пользователем)

2.1. Встроенные типы данных

Python имеет несколько встроенных типов данных:

1. Числовые типы

- int (целые числа) - неограниченной точности

- float (вещественные числа с плавающей точкой)

- complex (комплексные числа)

Примеры:

```
a = 42          # int
b = 3.14        # float
c = 2 + 3j      # complex
```

2. Строковые типы

- str (строки) - неизменяемые последовательности символов

Примеры:

```
name = "Hello"      # str
surname = 'World'   # str
multiline = """Это
многострочная
строка"""
```

3. Логический тип

- bool (логические значения) - True или False

Примеры:

```
is_valid = True
is_empty = False
```

4. Коллекции

- list (списки) - изменяемые последовательности

- tuple (кортежи) - неизменяемые последовательности

- dict (словари) - отображения ключ-значение

- set (множества) - неупорядоченные коллекции уникальных элементов

Примеры:

```
numbers = [1, 2, 3, 4, 5]           # list
coordinates = (10, 20)              # tuple
person = {"name": "John", "age": 30} # dict
unique_numbers = {1, 2, 3, 4, 5}    # set
```

2.2. Динамическая типизация

В Python переменные не имеют фиксированного типа. Тип определяется значением:

```
x = 42        # x имеет тип int
x = "hello"   # x теперь имеет тип str
x = [1, 2, 3] # x теперь имеет тип list
```

2.3. Проверка типов

Для проверки типа переменной используется функция type() или isinstance():

```
x = 42
print(type(x))              # <class 'int'>
print(isinstance(x, int))   # True
```

2.4. Преобразование типов

Python поддерживает неявное и явное преобразование типов:

```
# Неявное преобразование
result = 5 + 3.14  # int + float = float
# Явное преобразование
x = int("42")      # str -> int
y = str(42)        # int -> str
z = float("3.14")  # str -> float
```

2.5. Специальные типы

- None - специальное значение, означающее "ничего"
- bytes - неизменяемые последовательности байтов
- bytearray - изменяемые последовательности байтов

Примеры:

```
empty = None
data = b"hello"           # bytes
mutable_data = bytearray(b"hello")  # bytearray
```

2.6. Пользовательские типы

В Python можно создавать собственные типы с помощью классов:

```
class Person:
def __init__(self, name, age):
self.name = name
self.age = age
person = Person("John", 30)
print(type(person))  # <class '__main__.Person'>
```

2.7. Особенности типов в Python

1. Все в Python является объектом

2. Переменные являются ссылками на объекты

3. Неизменяемые типы (int, str, tuple) не могут быть изменены после создания

4. Изменяемые типы (list, dict, set) могут быть изменены после создания

Пример:

```
# Неизменяемый тип
x = "hello"
# x[0] = 'H'  # Ошибка! Строки неизменяемы
# Изменяемый тип
numbers = [1, 2, 3]
numbers[0] = 10  # OK! Списки изменяемы
```

Контрольные вопросы:

1. Какие основные типы данных есть в Python?
2. В чем отличие динамической типизации от статической?
3. Как проверить тип переменной в Python?
4. Какие типы являются неизменяемыми, а какие изменяемыми?
5. Как создать пользовательский тип в Python?
6. Что такое None в Python?
7. В чем разница между list и tuple?
8. Как работает преобразование типов в Python?

Лекция 3

Среда разработки Python и основы программирования

Цель лекции: ознакомиться с интегрированными средами разработки для Python, понять процедуру запуска и работы с этими системами. Уяснить что такое консольное приложение, как с ним работать. Изучить структуру папок, процесс выполнения, язык программы, основные алгоритмические структуры.

3.1 Основные характеристики среды разработки Python.

Современные IDE для Python позволяют создавать, выполнять, тестировать и редактировать проект в единой среде программирования. Python поставляется с базовой средой IDLE, но для профессиональной разработки рекомендуется использовать более мощные IDE.

Python является интерпретируемым языком высокого уровня, что означает:

- Высокопроизводительный интерпретатор Python с компиляцией в байт-код;
- Объектно-ориентированная модель программирования (все в Python является объектами);
- Простота синтаксиса и читаемость кода;
- Мощный отладчик для поиска и устранения ошибок;
- Богатая стандартная библиотека и экосистема пакетов.

Язык программирования Python предназначен для:

- профессионалов - разработчиков информационных систем;
- пользователей - для быстрого решения своих задач;

- ученых и аналитиков данных;
- веб-разработчиков;
- специалистов по машинному обучению и искусственному интеллекту.

Запуск Python осуществляется несколькими способами:

- Через командную строку: python или python3
- Через IDE (PyCharm, VS Code, Spyder и др.)
- Через Jupyter Notebook для интерактивной разработки
- Через IDLE (встроенная среда Python)

Основные компоненты IDE для Python:

- Строка заголовка с именем IDE
- Строка главного меню IDE
- Панели инструментов для быстрого выполнения часто используемых команд

В центре могут размещаться основные окна (на вкладках, если их несколько):

- Редакторы кода
- Консоль Python
- Браузер файлов
- Свойства проекта
- Слева размещается панель навигации по проекту
- Справа размещаются:
- Обозреватель файлов
- Окно переменных и отладки
- Внизу размещается окно «Терминал» или «Консоль Python»
- Окно «Список ошибок»

3.2 Установка и настройка Python

Python можно скачать с официального сайта python.org. Рекомендуется устанавливать последнюю стабильную версию (на момент написания - Python 3.11+).

После установки Python доступен через командную строку. Для проверки установки выполните:

```
python --version
```

или

```
python3 --version
```

Для работы с пакетами Python используется менеджер пакетов pip:

```
pip install package_name
```

Рекомендуется использовать виртуальные окружения для изоляции проектов:

```
 python -m venv myproject_env
# Активация (Windows)
 myproject_env\Scripts\activate
# Активация (Linux/Mac)
 source myproject_env/bin/activate
```

3.3 Создание консольного приложения

Консольное приложение создается простым созданием файла с расширением .py. Например, создайте файл hello.py:

```
1    # Простейшая программа на Python
2    print("Hello, World!")
3    print("Добро пожаловать в мир Python!")
```

Рисунок 2.1 – Простые команды

```
Hello, World!
Добро пожаловать в мир Python!
```

Рисунок 2.1 – Результат простой команды

Для запуска программы:

```
python hello.py
```

Консольное приложение позволяет:

- Использовать простой текстовый интерфейс
- Быстро тестировать алгоритмы
- Работать на любых системах где установлен Python
- Легко отлаживать код

3.4 Структура проекта Python

Проект Python обычно организуется в следующей структуре:

```
myproject/
├── main.py                # Основной файл программы
├── requirements.txt       # Список зависимостей
├── README.md              # Описание проекта
├── src/                   # Исходный код
│   ├── __init__.py
│   ├── module1.py
│   └── module2.py
├── tests/                 # Тесты
│   ├── __init__.py
│   └── test_module1.py
├── data/                  # Данные
└── docs/                  # Документация
```

3.5 Алфавит языка Python

Алфавит языка Python составляют символы таблицы кодов Unicode. Алфавит Python включает в себя:

1. Буквы латинского алфавита:

A…Z – 26 заглавных букв,

a...z – 26 строчных букв, (строчные и прописные буквы различаются)

_ - знак подчеркивания.

2. Строчные и прописные буквы кириллицы (можно использовать в идентификаторах).

3. Арабские цифры:

0...9 – 10 цифр.

4. Специальные символы:

- знаки арифметических операций

+ - * / // % **

- отношения

< > == != <= >=

- знаки пунктуации

. : ; , ' "

- скобки

( ) [ ] { }

- символы

@ # ^ & | ~

5. Составные символы:

== - равно,

!= - не равно,

<= - меньше или равно,

>= - больше или равно,

** - возведение в степень,

// - целочисленное деление,

+= -= *= /= - операторы присваивания

3.6 Словарь языка Python

Неделимые последовательности знаков образуют слова. Алфавит Python служит для построения слов, которые называются лексемами.

Лексемы (слова) подразделяются на:

- ключевые (зарезервированные) слова,
- встроенные функции и типы,
- идентификаторы пользователя,
- знаки (символы) операций;
- литералы;
- разделители.

Лексемы обособляются разделителями. Этой же цели служит множество пробельных символов, табуляция, символ новой строки и комментарии.

Комментарии – это пояснительный текст, который интерпретатором игнорируется. В Python определены комментарии:

- однострочный комментарий

""" - многострочный комментарий (тройные кавычки)

''' - многострочный комментарий (тройные апострофы)

Примеры:

```
# Это однострочный комментарий

x = 5  # Комментарий в конце строки

"""
Это многострочный комментарий
Может занимать несколько строк
"""
```

Идентификаторы (имена)

В языке Python используется кодировка Unicode. Это означает:

- Чувствительность к регистру, M и m - это разные переменные.
- Допустимо использовать для идентификаторов символы кириллицы.

На имена языка накладываются ограничения:

- Первый символ – обязательно буква или подчеркивание.
- Следующие символы – буквы, цифры, символ подчеркивания.

В языке Python для многословных имен рекомендуется использовать snake_case (слова разделяются подчеркиваниями) или PascalCase для классов.

Примеры правильных идентификаторов:

```
my_variable
MyClass
переменная_на_русском
_private_variable
CONSTANT_VALUE
```

Ключевые слова — это предварительно определенные зарезервированные идентификаторы, имеющие специальные значения для интерпретатора.

Примеры ключевых слов: and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield, True, False, None

Встроенные функции и типы

Python имеет множество встроенных функций: print(), input(), len(), type(), int(), str(), float(), list(), dict(), set(), tuple() и многие другие.

3.7 Переменные и константы в Python

Python является языком с динамической типизацией, что означает, что тип переменной определяется автоматически при присваивании значения.

Переменная – это синтаксическая единица, значение которой может изменяться. В Python переменные создаются при первом присваивании значения.

Формат объявления переменной:

```
имя_переменной = значение
```

Примеры:

```
x = 10  # целочисленная переменная
y = 3.14  # вещественная переменная
name = "Python"  # строковая переменная
is_active = True  # логическая переменная
numbers = [1, 2, 3, 4, 5]  # список
```

Рисунок 2.3 – Примеры представления перемынных

Константы в Python

В Python нет встроенной поддержки констант, но по соглашению константы записываются заглавными буквами:

```
SPEED_LIMIT = 55
PI = 3.14159265358979323846264338327950
```

Для создания настоящих констант можно использовать модуль enum или классы.

3.8 Структура программы Python

Программа на Python может быть как простым скриптом, так и сложным модулем с классами и функциями.

Простейшая программа:

```
# Простейшая программа на Python
print("Hello, World!")
```

```
1  # Простейшая программа на Python
2  print("Hello, World!")
```

Рисунок 2.4 – простейшая программа

```
Hello, World!
```

Рисунок 2.5 -Результат простейшей программы

Программа с функциями:

```
def greet(name):
"""Функция приветствия"""
return f"Привет, {name}!"
def main():
"""Главная функция"""
name = input("Введите ваше имя: ")
message = greet(name)
print(message)
if __name__ == "__main__":
main()
```

```
1  def greet(name):  1 usage
2      return f"Привет, {name}!"
3  def main():  1 usage
4      name = input("Введите ваше имя: ")
5      message = greet(name)
6      print(message)
7  if __name__ == "__main__":
8              main()
```

Рисунок 2.6 – Пример программы с функцией

```
Введите ваше имя: вылрпвы
Привет, вылрпвы!
```

Рисунок 2.7 - Результат программы с функцией

3.9 Базовые алгоритмические структуры

Алгоритм любой сложности может быть представлен комбинацией трёх базовых структур:

- следование;
- ветвление;
- повторение (цикл).

Структура "следование" означает, что несколько операторов должны быть выполнены последовательно друг за другом и только один раз за время выполнения данной программы.

Пример структуры следования:

```
# Линейный алгоритм
a = 5
b = 3
sum_result = a + b
print(f"Сумма: {sum_result}")
```

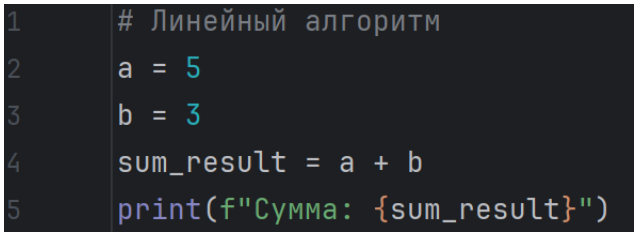

Рисунок 2.8 – Пример линейного алгоритма

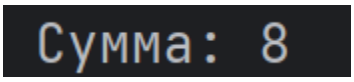

Рисунок 2.9 – результат

Структура "ветвление" разделяет последовательность действий на 2 направления в зависимости от итога заданного условия.

Пример ветвления:

```
# Условный оператор
age = int(input("Введите возраст: "))
if age >= 18:
print("Совершеннолетний")
else:
print("Несовершеннолетний")
# Тернарный оператор
```

```
status = "взрослый" if age >= 18 else "ребенок"
```

```python
# Условный оператор
age = int(input("Введите возраст: "))
if age >= 18:
    print("Совершеннолетний")
else:
    print("Несовершеннолетний")
# Тернарный оператор
status = "взрослый" if age >= 18 else "ребенок"
```

Рисунок – 2.10 – Пример ветвления

```
Введите возраст: 18
Совершеннолетний
```

Рисунок – 2.11 Результат 1

```
Введите возраст: 17
Несовершеннолетний
```

Рисунок 2.12 - Результат 2

Структура "повторение" обеспечивает повторное выполнение одного или группы операторов – цикл.

```
Примеры циклов:
# Цикл for
for i in range(5):
print(f"Итерация {i}")
```

```python
# Цикл for
for i in range(5):
    print(f"{i}")
```

Рисунок 2.13 – Цикл for

```
0
1
2
3
4
```

Рисунок 2.14 – Результат for

```
# Цикл while
count = 1
while count < 5:
print(f"Счетчик: {count}")
```

```
1  # Цикл while
2  count = 1
3  while count < 5:
4      print(f"Счетчик: {count}")
```

Рисунок 2.15 – Пример While

```
Счетчик: 1
Счетчик: 1
Счетчик: 1
Счетчик: 1
Счетчик: 1
Счетчик: 1
Счетчик: 1
Счетчик: 1
```

Рисунок 2.16 – Результат While

```
# Цикл с условием
numbers = [1, 2, 3, 4, 5]
for num in numbers:
if num % 2 == 0:
print(f"Четное число: {num}")
```

```
1  # Цикл с условием
2  numbers = [1, 2, 3, 4, 5]
3  for num in numbers:
4      if num % 2 == 0:
5          print(f"Четное число: {num}")
```

Рисунок 2.17 – Пример цикл с условием

```
Четное число: 2
Четное число: 4
```

Рисунок 2.18 – Результат цикла с условием

3.10 Методы алгоритмизации задач

Реальные алгоритмы представляют собой совокупность всех рассмотренных базовых структур. Наиболее часто употребляются:

- линейные вычисления,
- ветвления,
- выбор из большого количества альтернатив,
- циклы (вложенные циклы),
- функции и модули.

Пример комплексного алгоритма:

```
def find_primes(limit):
"""Поиск простых чисел до заданного предела"""
primes = []
for num in range(2, limit + 1):
is_prime = True
for i in range(2, int(num ** 0.5) + 1):
if num % i == 0:
is_prime = False
break
if is_prime:
primes.append(num)
return primes
# Использование функции
limit = int(input("Введите предел: "))
primes = find_primes(limit)
print(f"Простые числа до {limit}: {primes}")
```

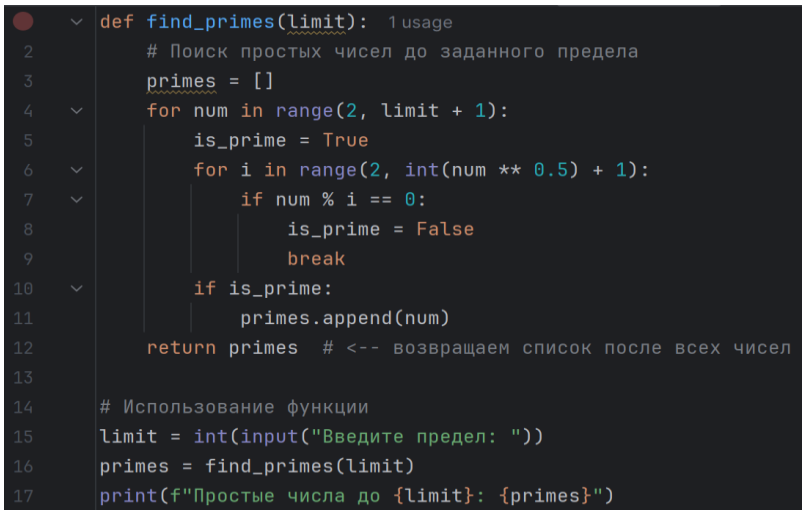

Рисунок 2.19 – пример комплексного алгоритма

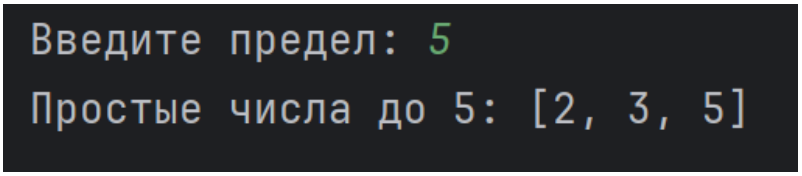

Рисунок 2.20 Результат комплексного алгоритма

Краткие итоги. В лекции были рассмотрены следующие вопросы: среды разработки для Python, работа консольного приложения, алфавит языка Python, базовые алгоритмические структуры и методы алгоритмизации задач.

Контрольные вопросы

1. Консольное приложение Python его работа и назначение.
2. Опишите процесс выполнения Python программы.
3. Алфавит языка Python его составляющие.
4. Дайте определение «лексемам», «идентификаторам», «ключевым словам», «переменные».
5. Опишите структуру программы Python.
6. Перечислите базовые алгоритмические структуры.
7. Какие IDE рекомендуются для разработки на Python?
8. В чем особенности динамической типизации в Python?
9. Что такое виртуальные окружения и зачем они нужны?
10. Какие встроенные функции Python вы знаете?

Лекция 4

Операторы в Python

Цель лекции: ознакомиться с понятием «операторы», как они применяются в программах на языке Python. Уяснить, для чего нужны операторы.

Оператор – это символ или ключевое слово, которое выполняет определенную операцию с одним или несколькими операндами. Операторы используются для выполнения вычислений, присваивания значений, проверки на равенство и неравенство и т. д.

В языке Python имеется большой набор операторов. Они представляют собой символы, определяющие операции, которые необходимо выполнить с выражением.

Операторы в выражениях исполняются с приоритетами:

- высший приоритет имеют основные операторы (скобки, вызовы функций)
- далее арифметические операторы (степень, умножение, деление)
- затем аддитивные (сложение, вычитание)
- далее операторы сравнения
- затем логические операторы

4.1 Арифметические операторы

Оператор	Описание	Пример
+	Сложение	a + b

-	Вычитание	a – b
*	Умножение	a * b
/	Деление (вещественное)	a / b
//	Целочисленное деление	a // b
%	Остаток от деления	a % b
**	Возведение в степень	a ** b

Таблица 4.1 - Арифметические операторы Python

Пример 1:

```python
a = 10
b = 3
print(f"a + b = {a + b}")     # 13
print(f"a - b = {a - b}")     # 7
print(f"a * b = {a * b}")     # 30
print(f"a / b = {a / b}")     # 3.333...
print(f"a // b = {a // b}")   # 3
print(f"a % b = {a % b}")     # 1
print(f"a ** b = {a ** b}")   # 1000
```

4.2 Операторы сравнения

Оператор	Описание	Пример
==	Равно	a == b
!=	Не равно	a != b
<	Меньше	a < b
>	Больше	a > b
<=	Меньше или равно	a <= b
>=	Больше или равно	a >= b
is	Тот же объект	a is b
is not	Не тот же объект	a is not b

Таблица 4.2 - Операторы сравнения Python

Пример 2:

```python
```

```
x = 5
y = 10
print(f"x == y: {x == y}")  # False
print(f"x != y: {x != y}")  # True
print(f"x < y: {x < y}")    # True
print(f"x > y: {x > y}")    # False
```

4.3 Логические операторы

Оператор	Описание	Пример
and	Логическое И	a and b
or	Логическое ИЛИ	a or b
not	Логическое НЕ	not a

Таблица 4.3 - Логические операторы Python

Пример 3:

```python
a = True
b = False
print(f"a and b: {a and b}")  # False
print(f"a or b: {a or b}")    # True
print(f"not a: {not a}")      # False
```

4.4 Операторы присваивания

Оператор	Описание	Пример	Эквивалент
=	Присваивание	a = b	a = b
+=	Сложение с присваиванием	a += b	a = a + b
-=	Вычитание с присваиванием	a -= b	a = a - b
*=	Умножение с присваиванием	a *= b	a = a * b
/=	Деление с присваиванием	a /= b	a = a / b
//=	Целочисленное деление с присваиванием	a //= b	a = a // b
%=	Остаток с присваиванием	a %= b	a = a % b
**=	Степень с присваиванием	a **= b	a = a ** b

Таблица 4.4 - Операторы присваивания Python

Пример 4:

```python
x = 10
x += 5    # x = x + 5 = 15
x *= 2    # x = x * 2 = 30
x //= 3   # x = x // 3 = 10
print(f"x = {x}")  # 10
```

4.5 Операторы принадлежности

Оператор	Описание	Пример
in	Принадлежит	x in y
not in	Не принадлежит	x not in y

Таблица 4.5 - Операторы принадлежности Python

Пример 5:

```python
numbers = [1, 2, 3, 4, 5]
print(f"3 in numbers: {3 in numbers}")        # True
print(f"6 in numbers: {6 in numbers}")        # False
print(f"6 not in numbers: {6 not in numbers}") # True
```

4.6 Операторы тождественности

Операторы is и is not проверяют, ссылаются ли две переменные на один и тот же объект в памяти.

Пример 6:

```python
a = [1, 2, 3]
b = [1, 2, 3]
c = a
print(f"a is b: {a is b}")      # False (разные
объекты)
print(f"a is c: {a is c}")      # True (один объект)
print(f"a == b: {a == b}")      # True (одинаковое
содержимое)
```

4.7 Побитовые операторы

Оператор	Описание	Пример
&	Побитовое И	a & b
|	Побитовое ИЛИ	a | b
^	Побитовое исключающее ИЛИ	a ^ b
~	Побитовое НЕ	~a
<<	Сдвиг влево	a << b
>>	Сдвиг вправо	a >> b

Таблица 4.6 - Побитовые операторы Python

Пример 7:

```python
a = 5  # 101 в двоичном
b = 3  # 011 в двоичном
print(f"a & b = {a & b}")  # 001 = 1
print(f"a | b = {a | b}")  # 111 = 7
print(f"a ^ b = {a ^ b}")  # 110 = 6
print(f"~a = {~a}")        # -6
print(f"a << 1 = {a << 1}") # 10
print(f"a >> 1 = {a >> 1}") # 2
```

4.8 Приоритет операторов

Операторы выполняются в следующем порядке (от высшего к низшему приоритету):

1. ** (степень)
2. +x, -x, ~x (унарные)
3. *, /, //, % (мультипликативные)
4. +, - (аддитивные)
5. <<, >> (сдвиги)
6. & (побитовое И)
7. ^ (побитовое исключающее ИЛИ)
8. | (побитовое ИЛИ)
9. ==, !=, <, >, <=, >=, is, is not, in, not in (сравнения)

10. not (логическое НЕ)
11. and (логическое И)
12. or (логическое ИЛИ)

Пример 8:

```python
result = 2 + 3 * 4 ** 2  # 2 + 3 * 16 = 2 + 48 = 50
print(f"result = {result}")
# Использование скобок для изменения приоритета
result = (2 + 3) * 4 ** 2  # 5 * 16 = 80
print(f"result with parentheses = {result}")
```

4.9 Специальные операторы Python

1. Оператор моржа (:=) - Python 3.8+

   Позволяет присваивать значение переменной в выражении:

   ```python
   # Обычный способ
   n = len([1, 2, 3])
   if n > 2:
   print(f"Длина списка: {n}")
   # С оператором моржа
   if (n := len([1, 2, 3])) > 2:
   print(f"Длина списка: {n}")
   ```

2. Тернарный оператор

   Условное выражение в одну строку:

   ```python
   age = 20
   status = "взрослый" if age >= 18 else "несовершеннолетний"
   print(status)  # взрослый
   ```

Контрольные вопросы:

1. Какие арифметические операторы есть в Python?
2. В чем разница между операторами / и //?
3. Как работают логические операторы and, or, not?

4. Что такое операторы принадлежности in и not in?
5. В чем разница между == и is?
6. Как изменить приоритет выполнения операторов?
7. Что такое тернарный оператор в Python?
8. Как работают побитовые операторы?

Лекция 5

Инструкции управления в Python

Цель лекции: разобрать инструкцию if, где она применяется и как работает. Изучить вложенные инструкции, инструкции выбора, как они работают.

5.1 Инструкция if

Применяется для ветвления по двум ветвям.

Полный формат (синтаксис) инструкции:

```python
if условие:
инструкция1    # одна инструкция языка
else:
инструкция2    # одна инструкция языка
```

Если условие выполняется, то исполняется инструкция1, в противном случае исполняется инструкция2.

Фраза else может отсутствовать, т.к. это необязательная часть инструкции if.

В общем виде, когда надо выполнить несколько действий, инструкция if записывается следующим образом:

```python
if условие:
# Блок инструкций 1 - несколько действий
инструкция1
инструкция2
инструкция3
else:
# Блок инструкций 2 - несколько действий
инструкция4
инструкция5
инструкция6
```

Фраза else может отсутствовать, т.к. это необязательная часть инструкции if.

Если условие выполняется, то исполняется Блок инструкций 1, в противном случае исполняется Блок инструкций 2.

Выполняется инструкция if следующим образом:

1. Вычисляется значение условия (True или False)

2. Если условие истинно (True), то выполняются инструкции1. На этом выполнение инструкций if завершается, инструкции, следующие за словом else, не будут выполнены

3. Если условие ложно (False), то выполняются инструкции, следующие за словом else, а инструкции за словом if игнорируются

Пример 1. Поиск максимального числа из двух чисел x и y.

```python
# 1-й вариант решения
x = 12
y = 5
if x > y:
max_val = x
else:
max_val = y
print(f"Максимальное: {max_val}")
# 2-й вариант решения (без else)
x = 12
y = 5
max_val = x
if y > x:
max_val = y
print(f"Максимальное: {max_val}")
```

5.2 Вложенные инструкции if

Внутри блока if можно размещать другие инструкции if. Это называется вложенными инструкциями.

Пример 2. Определение знака числа.

```python
```

```
number = int(input("Введите число: "))
if number > 0:
print("Число положительное")
else:
if number < 0:
print("Число отрицательное")
else:
print("Число равно нулю")
```

5.3 Инструкция elif

Для упрощения записи вложенных if-else используется инструкция elif (else if):

```python
if условие1:
блок_инструкций1
elif условие2:
блок_инструкций2
elif условие3:
блок_инструкций3
else:
блок_инструкций4
```

Пример 3. Определение времени суток.

```python
hour = int(input("Введите час (0-23): "))
if 6 <= hour < 12:
print("Утро")
elif 12 <= hour < 18:
print("День")
elif 18 <= hour < 22:
print("Вечер")
else:
print("Ночь")
```

5.4 Инструкция if без else

Если else не нужен, его можно опустить:

```python
age = int(input("Введите возраст: "))
if age >= 18:
print("Вы совершеннолетний")
print("Программа завершена")
```

```

### 5.5 Тернарный оператор

Для простых условий можно использовать тернарный оператор:

```python
# Обычная запись
x = 10
y = 5
if x > y:
max_val = x
else:
max_val = y
# Тернарный оператор
max_val = x if x > y else y
```

5.6 Логические операторы в условиях

В условиях можно использовать логические операторы and, or, not:

```python
age = int(input("Введите возраст: "))
has_license = input("Есть ли водительские права? (да/нет): ").lower() == "да"
if age >= 18 and has_license:
print("Можете водить автомобиль")
elif age >= 16 and has_license:
print("Можете водить скутер")
else:
print("Не можете водить")
```

5.7 Операторы сравнения в условиях

```python
score = int(input("Введите балл (0-100): "))
if score >= 90:
grade = "A"
elif score >= 80:
grade = "B"
elif score >= 70:
grade = "C"
elif score >= 60:
grade = "D"
else:
grade = "F"
```
```

```
print(f"Ваша оценка: {grade}")
```

5.8 Проверка принадлежности

```python
fruits = ["яблоко", "банан", "апельсин"]
fruit = input("Введите фрукт: ")
if fruit in fruits:
print(f"{fruit} есть в списке")
else:
print(f"{fruit} нет в списке")
```

5.9 Проверка типа данных

```python
value = input("Введите значение: ")
if value.isdigit():
print("Это число")
elif value.isalpha():
print("Это строка из букв")
elif value.isalnum():
print("Это строка из букв и цифр")
else:
print("Это специальные символы")
```

5.10 Множественные условия

```python
temperature = int(input("Введите температуру: "))
humidity = int(input("Введите влажность (0-100): "))
if temperature > 30 and humidity > 70:
print("Жарко и влажно")
elif temperature > 30 and humidity <= 70:
print("Жарко, но не влажно")
elif temperature <= 30 and humidity > 70:
print("Прохладно и влажно")
else:
print("Прохладно и сухо")
```

5.11 Сложные логические выражения

```python
age = int(input("Введите возраст: "))
income = int(input("Введите доход: "))
```

```
    has_credit = input("Есть ли кредитная история? (да/нет): ").lower() == "да"
    if (age >= 21 and income >= 50000) or (age >= 25 and has_credit):
    print("Кредит одобрен")
    else:
    print("Кредит не одобрен")
    ```
```

### 5.12 Обработка исключений в условиях

```
    ```python
    try:
    number = int(input("Введите число: "))
    if number > 0:
    print("Положительное число")
    elif number < 0:
    print("Отрицательное число")
    else:
    print("Ноль")
    except ValueError:
    print("Ошибка: введено не число")
    ```
```

**Контрольные вопросы:**

1. Как работает инструкция if в Python?
2. В чем разница между if и if-else?
3. Что такое вложенные инструкции if?
4. Для чего используется инструкция elif?
5. Что такое тернарный оператор?
6. Как использовать логические операторы в условиях?
7. Как проверить принадлежность элемента списку?
8. Как обработать исключения в условиях?
```

Лекция 6

Циклы в Python

Цель лекции: изучить решение циклической формы с помощью языка Python. Рассмотреть операторы цикла, их назначение и применение. Дать определение цикла.

6.1 Операторы цикла

Алгоритмы решения многих задач являются циклическими, т.е. для достижения результата определенная последовательность действий должна быть выполнена несколько раз. Циклом называется группа инструкций, повторяющихся многократно с разными данными.

В Python для циклов применяются инструкции: for, while, а также специальные конструкции для работы с итерируемыми объектами.

6.2 Цикл for

Цикл for используется для итерации по последовательностям (списки, строки, кортежи, словари и т.д.).

Синтаксис:

```python
for переменная in последовательность:
блок_инструкций
```

Пример 1. Вывод последовательности чисел.

```python
# Вывод чисел от 0 до 9
for i in range(10):
print(f"i = {i}")
# Вывод элементов списка
fruits = ["яблоко", "банан", "апельсин"]
for fruit in fruits:
```

```
print(fruit)
```

6.3 Функция range()

Функция range() создает последовательность чисел:

```python
range(stop)                    # от 0 до stop-1
range(start, stop)             # от start до stop-1
range(start, stop, step)        # от start до stop-1 с шагом step
```

Примеры:

```python
# range(5) -> 0, 1, 2, 3, 4
for i in range(5):
print(i)
# range(2, 7) -> 2, 3, 4, 5, 6
for i in range(2, 7):
print(i)
# range(0, 10, 2) -> 0, 2, 4, 6, 8
for i in range(0, 10, 2):
print(i)
```

6.4 Цикл while

Цикл while повторяет блок инструкций, пока условие истинно.

Синтаксис:

```python
while условие:
блок_инструкций
```

Пример 2. Вычисление значений функций.

```python
import math
# Параметры
a = 1.3
b = 1.29
tn = 0.1
tk = 2.2
h = 0.3
print("Таблица расчета функции s")
print()
```

```
t = tn
while t <= tk:
x = a * t
if x < 1:
s = x + b
elif x == 1:
s = math.cos(x)
else:
s = math.exp(x) * math.cos(x)
print(" ********************** ")
print(f" * x= {x:4.1f} * s= {s:6.3f} *")
t += h
```

6.5 Вложенные циклы

Циклы могут быть вложенными друг в друга.

Пример 3. Таблица умножения.

```python
# Таблица умножения 5x5
for i in range(1, 6):
for j in range(1, 6):
print(f"{i} x {j} = {i * j}", end="\t")
print()  # Переход на новую строку
```

6.6 Управление циклами

1. break - прерывание цикла

```python
# Поиск первого четного числа
numbers = [1, 3, 5, 8, 9, 12]
for num in numbers:
if num % 2 == 0:
print(f"Первое четное число: {num}")
break
```

2. continue - пропуск итерации

```python
# Вывод только нечетных чисел
for i in range(10):
if i % 2 == 0:
continue
print(f"Нечетное число: {i}")
```

3. else в циклах

Блок else выполняется, если цикл завершился нормально (без break):

```python
# Поиск простого числа
n = 17
for i in range(2, n):
if n % i == 0:
print(f"{n} не является простым числом")
break
else:
print(f"{n} - простое число")
```

6.7 Итерация по словарям

```python
person = {"имя": "Иван", "возраст": 25, "город":
"Москва"}
# Итерация по ключам
for key in person:
print(f"{key}: {person[key]}")
# Итерация по ключам и значениям
for key, value in person.items():
print(f"{key}: {value}")
# Итерация только по значениям
for value in person.values():
print(value)
```

6.8 Итерация по строкам

```python
text = "Python"
for char in text:
print(char)
# Итерация с индексами
for i, char in enumerate(text):
print(f"Индекс {i}: {char}")
```

6.9 Генераторы списков (List Comprehensions)

```python
# Создание списка квадратов
squares = [x**2 for x in range(10)]
print(squares)  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
# Создание списка четных чисел
even_numbers = [x for x in range(20) if x % 2 == 0]
print(even_numbers)  # [0, 2, 4, 6, 8, 10, 12, 14, 16,
18]
# Создание словаря из списка
words = ["hello", "world", "python"]
word_lengths = {word: len(word) for word in words}
print(word_lengths)  # {'hello': 5, 'world': 5,
'python': 6}
```

6.10 Циклы с условиями

```python
# Сумма чисел до ввода 0
total = 0
while True:
number = int(input("Введите число (0 для выхода): "))
if number == 0:
break
total += number
print(f"Сумма: {total}")
```

6.11 Обработка исключений в циклах

```python
numbers = ["1", "2", "abc", "4", "5"]
valid_numbers = []
for num_str in numbers:
try:
num = int(num_str)
valid_numbers.append(num)
except ValueError:
print(f"'{num_str}' не является числом")
print(f"Валидные числа: {valid_numbers}")
```

6.12 Практические примеры

Пример 4. Поиск максимального элемента в списке.

```python
numbers = [3, 7, 2, 9, 1, 5]
max_num = numbers[0]
for num in numbers[1:]:
if num > max_num:
max_num = num
print(f"Максимальное число: {max_num}")
```

Пример 5. Подсчет символов в строке.

```python
text = "Hello, World!"
char_count = {}
for char in text:
if char in char_count:
char_count[char] += 1
else:
char_count[char] = 1
for char, count in char_count.items():
print(f"'{char}': {count}")
```

Пример 6. Числа Фибоначчи.

```python
n = 10
fib = [0, 1]
for i in range(2, n):
fib.append(fib[i-1] + fib[i-2])
print(f"Первые {n} чисел Фибоначчи: {fib}")
```

6.13 Оптимизация циклов

```python
# Медленный способ
result = []
for i in range(1000):
if i % 2 == 0:
result.append(i**2)
# Быстрый способ (генератор списка)
result = [i**2 for i in range(1000) if i % 2 == 0]
# Еще быстрее (генератор)
result = (i**2 for i in range(1000) if i % 2 == 0)
```

Контрольные вопросы:

1. В чем разница между циклами for и while?
2. Как работает функция range()?
3. Для чего используются break и continue?
4. Когда выполняется блок else в циклах?
5. Как итерироваться по словарям?
6. Что такое генераторы списков?
7. Как обработать исключения в циклах?
8. Как оптимизировать производительность циклов?

Лекция 7

Массивы (Списки) в Python

Цель лекции: дать определение, что такое список в Python. Изучить структуру списков, какие они бывают. Уяснить, как объявляется формат списка. Что такое инициализация. Рассмотреть примеры решения задач со списками.

Список (list) – структура данных, представляющая собой набор переменных, имеющих общее имя.

Каждый элемент списка однозначно определяется именем и индексом (номером элемента в списке).

Индексы списка принадлежат целочисленному типу. Списки позволят легко обрабатывать большое количество связанных переменных.

Список может быть:

- одномерным A1 [ ]
- двумерным A2 [[ ], [ ]] (список списков)

Список заданный математически имеет вид:

- одномерный - строка - A = [1, 5, 8, 3, 4, 2, 7, 0] или столбец
- двумерный B (например, 3 – строки и 2 – столбца)

Формат объявления одномерного списка в Python:

```
ИмяСписка = [элемент1, элемент2, ...]  # с инициализацией
ИмяСписка = []  # пустой список
ИмяСписка = [0] * размер  # список с нулевыми элементами
```

Формат объявления двумерного списка в Python:

```
ИмяСписка = [[элемент11, элемент12], [элемент21, элемент22]]  # с инициализацией
ИмяСписка = [[0] * столбцы for i in range(строки)]  # пустой двумерный список
ИмяСписка – имя списка.
```

Элементы списка могут быть любых типов (int, float, str, bool и даже другие списки).

Доступ к элементу списка осуществляется посредством индекса (позицию элемента внутри списка):

```
ИмяСписка [ НомерЭлемента ].
```

Индексация списков начинается с нуля: список с элементами n индексируется от 0 до n-1. При обращении к элементу списка, надо указать его имя и номер элемента в квадратных скобках.

Например: A[0] – первый элемент списка A.

A[4] – пятый элемент списка A.

B[2][3] – элемент, лежащий на пересечении 3 строки и 4 столбца списка B.

Примеры объявления списков:

```
# Одномерные списки
A = [0] * 5  # одномерный список A из 5 целых чисел
H = [''] * 10  # одномерный список H из 10 символов
M = [[0] * 3 for i in range(2)]  # двумерный список
2x3 вещественных
#чисел, содержащий 2 строки и 3 столбца
```

При объявлении списка можно выполнить его инициализацию, т.е. присвоить начальные значения элементам списка в момент его создания.

Формат объявления одномерного списка с инициализацией:

```
ИмяСписка = [v1, v2, v3, ...vn]
```

Формат объявления двумерного списка в Python:

```
ИмяСписка = [[v11, v12, ...v1j], [vi1, vi2, ...vij]]
#1-я строка …… i-я строка
```

Здесь начальные значения, присваиваемые элементам списка, задаются с помощью последовательности v1, v2, v3,…vn для одномерного списка и vij – для списка размерностью (i x j), где i – номер строки, а j – номер столбца для двумерного списка.

Примеры объявления списков с инициализацией:

```
C = [1, 3, 5, -7, 9]  # список C инициализирован пятью элементами
D = [1.5, 2.1, 3.65, 4.7, 5.14, 6.36]  # список вещественных чисел
Q = ['a', 'b', 'g']  # список Q инициализирован тремя символами
Team = ["Zenith", "Dynamo", "Sparta", "Rotor", "CSK"]
M = [[1, 2, 3], [4, 5, 6]]  # список M: 2 строки и 3 столбца инициализирован
L = [[0, 2, 4, 6], [2, 9, 6, 3], [4, 7, 5, 8], [1, 6, 5, 7]]
#список L (4x4) инициализирован
```

Список можно инициализировать во время выполнения программы в цикле.

Например:

```
R = [0] * 5
for k in range(5):
print(f"Введите {k} элемент списка R")
R[k] = int(input())
```

В Python также предусмотрена инструкция for для итерации элементов списка.

Синтаксис инструкции for:

for элемент in ИмяСписка:

Инструкции тела цикла

элемент – переменная для элемента списка.

ИмяСписка – имя списка.

Пример. Следующий код создает список Числа и осуществляет его итерацию с помощью инструкции for.

```
Числа = [4, 5, 6, 1, 2, 3, -2, -1, 0]
for i in Числа:
print(i, end=" ")
```

Вывод в консоль строки: 4 5 6 1 2 3 -2 -1 0

Операции, производимые над элементами списка, полностью определяются типом этих элементов.

7.1 Операции со списками

Типовые операции при работе со списками:

- Вывод списка;
- Ввод списка;
- Поиск максимального/минимального элемента списка;
- Поиск заданного элемента списка;
- Сортировка списка.

Часто используемые операции: накопление суммы элементов списка, расчет среднего арифметического значения элементов списка.

Под ***выводом списка*** понимается вывод на экран монитора, значений элементов заданного / сформированного списка.

При выводе всех элементов списка удобно использовать оператор цикла for, при этом переменная счетчик может быть использована в качестве индекса элементов списка.

Пример вывода одномерного списка Team:

```
Team = ["Zenit", "Dynamo", "Sparta", "Rotor", "CSK"]
for i in Team:
print(i, end=" ")
```

Под ***вводом списка*** понимается процесс получения от пользователя во время работы программы, значений элементов списка.

При вводе элементов списка удобно использовать оператор цикла for, при этом переменная счетчик может быть использована в качестве индекса элементов списка.

Иногда в качестве элементов списка используют случайные числа, которые можно получить с помощью функции random.randint() - случайное число в заданном диапазоне.

Программа помещает в список А числа от 0 до 9.

```
import random
A = [0] * 10  # одномерный список А из 10 целых чисел
for i in range(10):
A[i] = i
for i in range(10):
print(f"A[{i}]={A[i]}")
```

Таблица 7.1

Схематично представленный список А

Пример заполнения двумерного списка Т[3х4] числами от 1 до 12:

```
import random
T = [[0] * 4 for i in range(3)]
print("Сформированная матрица")
print()
for i in range(3):
for j in range(4):
T[i][j] = (i * 4) + j + 1
print(f"{T[i][j]:2d}", end=" ")
print()
```

В этом примере элемент списка Т[0][0] примет значение 1, Т[0][1] примет значение 2 … Т[2][3] примет значение 12. Схематично список Т можно представить:

Поиск максимального (или минимального) элемента списка

Алгоритм поиска. Делается предположение, что 1-й элемент списка max (min), затем все элементы списка сравниваются с ним. Если во время проверки обнаруживается, что очередной элемент больше (меньше) принятого за max (min), то этот элемент становится максимальным (минимальным).

Пример 1. В списке X из 10 элементов вычислить наибольший элемент списка и его номер.

```
X = [-5, 6, -8, 2, 4, 9, -7, 4, 1, 0]  # одномерный список X из 10 целых чисел
Xmax = X[0]  # максимальный элемент
Imax = 0     # номер максимального элемента
for i in range(10):
print(f"X[{i}]={X[i]}")
print()  # вывод элементов списка
for i in range(1, 10):
if X[i] > Xmax:
Xmax = X[i]  # выявление max
Imax = i
print(f"Максимальный элемент: {Xmax}")
print(f"Его номер: {Imax + 1}")
```

При выявлении минимального элемента условие > заменяется на <:

```
if X[i] < Xmin:
Xmin = X[i]  # выявление min
Imin = i     # его номера
```

Поиск заданного элемента списка

Под поиском заданного элемента списка понимается необходимость определить, содержит ли список определенную информацию или нет. Наиболее простой алгоритм поиска – простой перебор неупорядоченных элементов.

Поиск осуществляется последовательным сравнением элементов списка с образцом до тех пор, пока не будет найден элемент, равный образцу, или пока не будут проверены все элементы.

Для поиска элементов используют операторы цикла for, while

Пример 2. Найти заданный элемент в списке и вывести его на экран дисплея.

```
Mas = [-5, 6, -8, 2, 4, 9, -7, 4, 1, 0]  # одномерный список Mas из 10 целых чисел
obr = int(input("Введите образец для поиска: "))  # образец для поиска
```

```
Yes = False  # признак обнаружения
for i in range(10):
print(f"Mas[{i}]={Mas[i]}")
print()
i = 0
while i < 10 and not Yes:
if Mas[i] == obr:
Yes = True
break
else:
i = i + 1
if Yes:
print(f"Имеется совпадение с элементом {Mas[i]}. Его индекс: {i}")
else:
print("Совпадение с образцом отсутствует")
```

Сортировка списка

Под сортировкой списка подразумевается процесс перестановки элементов списка, с целью размещения элементов списка в определенном порядке.

Например, для целых чисел A после сортировки по возрастанию должно выполняться условие:

A[0] <= A[1] <= A[2] <= ... <= A[size-1], где size – длина списка

Алгоритм сортировки:

1. Просмотреть список от 1 элемента, найти min элемент и поместить его на место 1 элемента, а 1-й на место min.
2. Просмотреть список от 2 элемента, найти min элемент и поместить его на место 2 элемента, а 2-й на место min
3. И так далее до последнего элемента.

Элементы списка A

```
A[0] = 2
A[1] = 6
A[2] = -5
A[3] = 3
A[4] = 20
A[5] = -10
```

```
A[6] = 8
A[7] = 0
A[8] = 9
A[9] = -2
```

Пример 3. Отсортировать список целых чисел по возрастанию.

```
A = [2, 6, -5, 3, 20, -10, 8, 0, 9, -2]  # одномерный
список A из 10 целых чисел
for i in range(10):
print(f"A[{i}]={A[i]}")
print()
print("Сортировка списка")
print()
for i in range(9):
min_idx = i
for j in range(i + 1, 10):
if A[j] < A[min_idx]:
min_idx = j
A[i], A[min_idx] = A[min_idx], A[i]  # обмен элементов
for k in range(10):
print(f" {A[k]}", end="")
print()
print()
print("Список отсортирован")
print()
for k in range(10):
print(f" {A[k]}", end="")
```

Расчет статистических показателей

Пример 4. Рассчитать суммарное и среднее значение элементов заданного списка.

```
N = [99, 10, 100, 18, 78, 23, 63, 9, 87, 49]  #
одномерный список A из 10 целых чисел
sum_val = 0  # суммарное значение
for i in range(10):
sum_val = sum_val + N[i]
avg = sum_val / 10
print(f"Сумма: {sum_val}")
print(f"Среднее: {avg}")
# Для двумерного списка
N = [[-99, 10, 50, 18], [78, -23, 63, 9], [87, 49, -
55, 10], [16, 8, 95, -16]]
# двумерный список N из 4x4 целых чисел
print("Матрица N[4,4] целых чисел")
print()  # Вывод матрицы
```

```
for i in range(4):
for j in range(4):
print(f"{N[i][j]:2d}   ", end="")
print()
print()
sum_val = 0  # суммарное значение
c = 0  # счетчик для подсчета количества элементов
for i in range(4):
for j in range(4):
sum_val = sum_val + N[i][j]
c = c + 1
avg = sum_val / c
print("Результаты")
print()
print(f"Сумма: {sum_val}")
print(f"Среднее: {avg}")
```

Использование генератора случайных чисел

Иногда в качестве элементов списка используют случайные числа, которые можно получить с помощью функции random.randint(), которая генерирует случайное число в заданном диапазоне.

Пример 5. Ввод одномерного списка A с использованием программного генератора случайных чисел.

```
import random
n = 0
N = 10  # Переменные типа int
print()
A = [0] * N  # Список типа int
print("Создан список A случайных целых чисел")
print()
print("Номер n" + " Значение A[n]")
for n in range(N):
A[n] = random.randint(0, 99)  # Генерация случайного
числа в диапазоне от 0 до 99
print(f" {n} {A[n]:10d}")
print()
print("Нажмите любую клавишу")
input()  # Пауза
```

Как видно генератор выдает числа в диапазоне от 0 до 99

Пример 6. Вычислить наименьший элемент Mmin списка M и его порядковый номер Nmin. Размерность списка N=10. В программе элементы списка создаются генератором случайных чисел.

```
import random
n = 0
N = 10
Nmin = 0
Mmin = 0  # Переменные типа int
print()
M = [0] * N  # Список типа int
print("Создан список M случайных целых чисел")
print()
print("Номер n" + " Значение M[n]")
for n in range(N):
M[n] = random.randint(0, 99)  # Генерация случайного
числа
print(f" {n} {M[n]:10d}")
Mmin = M[0]  # Предположение
Nmin = 0
for n in range(1, N):
if M[n] < Mmin:  # Обнаружение минимума
Mmin = M[n]
Nmin = n
print()
print("Результаты")
print()
print(f"Nmin={Nmin}")  # Вывод Nmin
print(f"Mmin={Mmin}")  # Вывод Mmin
print()
print("Нажмите любую клавишу")
input()  # Пауза
```

Пример 7. Вывести квадратную матрицу X размером RxC в виде таблицы. Найти максимальное значение элемента и его координаты: номер строки R и номер столбца C.

```
import random
print("Введите число строк R и столбцов матрицы")
R = int(input("Число строк R="))  # Введите R
C = int(input("Число столбцов C="))  # Введите C
print()
M = [[0] * C for i in range(R)]  # Матрица M[R,C] типа
int
Mmax = 0
```

```
Rmax = 0
Cmax = 0
for r in range(R):  # Внешний цикл по строкам
for c in range(C):  # Внутренний цикл по столбцам
Mm = random.randint(0, 99)  # Генерация случайного
числа от 0 до 99
M[r][c] = Mm
if M[r][c] > Mmax:  # Обнаружение максимума
Mmax = M[r][c]  # Значение максимума
Rmax = r  # Номер строки
Cmax = c  # Номер столбца
# Вывод матрицы
print("Создана матрица M[R,C] случайных целых чисел")
print()
for r in range(R):
for c in range(C):
print(f"{M[r][c]:2d}   ", end="")
print()
print()
print("Результаты")
print()
print(f"Максимальное значение Mmax={Mmax}")  # Вывод
Mmax
print(f"Номер строки Rmax={Rmax}")  # Вывод Rmax
print(f"Номер столбца Cmax={Cmax}")  # Вывод Cmax
input()  # Пауза
```

По результату нужно проверить правильность исполнения алгоритма.

Списки строк

Подобно другим типам данных строки могут быть собраны в списки.

Пример 8. В заданном строковом списке заменить одни слова на другие.

```
str_list = ["Это ", "очень ", "простой ", "тест."]
print("Исходный список: ")
print()
for i in range(len(str_list)):
print(str_list[i] + "  ", end="")
print("\n")  # Вывод на печать через клавишу Tab
# Изменяем строку
str_list[1] = "тоже "
```

```
str_list[3] = "тест, не правда ли?"
print("Модифицированный список: ")
print()
for i in range(len(str_list)):
print(str_list[i] + "  ", end="")
input()  # Пауза
```

Пример 9. Ввести список символов и заменить один из символов, например 'a' на другой, например 'x'. Подсчитать количество замен.

```
Q = [''] * 10
a = 'a'
x = 'x'
c = 0  # Количество замен
print("Ввод символов в список Q")
for k in range(10):
print(f"Введите {k} элемент списка Q")
Q[k] = input()
print()
print("Исходный список Q: ")
for k in Q:
print(f"{k} ", end="")  # вывод элементов списка Q
print()
for k in range(10):
if Q[k] == 'a':  # символ, который надо найти
Q[k] = 'x'  # символ, на который надо заменить
c = c + 1  # количество замен
print()
print("Модифицированный список Q: ")
for k in Q:
print(f"{k} ", end="")  # вывод элементов списка S
print()
print()
print(f"Количество замен = {c}")
input()  # Пауза
```

Итоги лекции. В лекции были представлены примеры решения задач со списками, как одномерными, так и двумерными. Было дано определение списка, изучены способы задания, вывод. Изучены форматы объявления списка как одномерного, так и двумерного, инструкция for. Использование генератора чисел при задании списка.

Контрольные вопросы

1. Что такое список в Python?

2. Описание типа - список.
3. Какие операторы языка можно использовать для описания списков?
4. Особенности организации цикла при обработке списков?
5. Особенности программирования при обработке списков?
6. Особенность ввода и вывода списков?
7. Отличие задания одномерного списка от двумерного.
8. Как создать двумерный список случайных целых чисел?
9. Как создать двумерный список вещественных чисел?
10. Каков формат объявления двумерного списка?
11. С чего начинается индексация списков?

Лекция 8

Символы и строки в Python

Цель лекции: рассмотреть понятие символа и строки, их различия. Как они задаются в программе. Для каких операций они применяются.

ПК может обрабатывать не только числовую информацию, но и символьную. Символьная информация может быть представлена:

- Отдельными символами
- Строками

8.1 Символы

В Python для хранения и обработки отдельных символов используются строки длиной в один символ.

Python использует кодировку Unicode, в которой каждый символ может быть представлен различным количеством байтов (обычно 1-4 байта).

К символам относятся:

- Буквы русского и латинского алфавитов;
- Цифры;
- Знаки препинания;
- Специальные символы (пробел, '\n' - "новая строка");

Объявление символьных данных:

- Имя_переменной = 'значение' # переменная с инициализацией
- Имя_переменной = "значение" # переменная с инициализацией (альтернативный синтаксис)

Пример1:

```
sim = ''  # Символьная переменная без инициализации (пустая строка)
q = '*'   # Символьная переменная с инициализацией
s = 'x'   # Символьная константа
```

Символьная переменная может получить значение в программе в результате выполнения операции присвоения.

Пример2:

```
C = ''  # переменные без инициализации
S = ''  # переменные без инициализации
# ... ... ... ...
C = '*'  # переменной C присваивается значение *
S = 'a'  # переменной S присваивается значение a
# ... ... ... ...
```

Пример 3. Выполняет программу диалога: "Вы хотите научиться программировать?"

```
    s = input("Вы хотите научиться программировать? Введите Y или N: ")
    print(f"Ваш ответ: {s}")
```

8.2 Строки

Последовательность символов произвольной длины называется строкой. В Python строки являются неизменяемыми объектами типа str.

Объявление строковых данных:

```
    Имя_переменной = ""   # переменная без инициализации (пустая строка)
    Имя_переменной = "значение"      # переменная с инициализацией
    Имя_переменной = 'значение'      # переменная с инициализацией (альтернативный синтаксис)
```

Пример 4:

```
s = ""  # Строковая переменная без инициализации
s1 = "Привет"  # Строковая переменная с инициализацией
s2 = "Я студент"  # Строковая константа
```

Строковая переменная может получить значение в программе в результате выполнения операции присвоения.

Пример 5:

```
s1 = ""  # переменные без инициализации
s2 = ""  # переменные без инициализации
# ... ... ... ...
s1 = "Я студент"  # переменной s1 присваивается
значение "Я студент"
s2 = "Привет от меня"  # переменной s2 присваивается
значение "Привет от меня"
# ... ... ... ...
```

Пример 6. Выполняет программу диалога: "Вы хотите научиться программировать?"

```
s = input("Вы хотите научиться программировать? ")
print(f"Ваш ответ: {s}")
```

Результат выполнения строкового выражения является строка символов.

Для строк применимы операции:

```
конкатенация (добавление к первой строке второй)   # Сцепление строк
сравнение.  # По длине строки
```

Пример 7. Объединение (конкатенация) двух строк.

```
c = "C#"
h = "Express"
q = '!'
print(f"Введите 1 строку: {c}")
print(f"Введите 2 строку: {h}")
ch = c + h + q
print()
print(f"Ответ: {ch}")
Пример 8. Сравнить содержимое двух строк
s1 = input("Введите 1 строку: ")
print()
s2 = input("Введите 2 строку: ")
print()
if s1 == s2:
print("Строки равные")
else:
print("Строки не равные")
```

Переменную типа str можно сравнивать с другой переменной типа str.

Строки сравниваются посимвольно, начиная с 1 символа:

Если все символы сравниваемых строк одинаковые, то такие строки считаются равными, в противном случае – не равными. Для равенства двух строк применяют операторы == или !=.

Операторы отношения >, <, >=, <= с переменными типа str работают по лексикографическому порядку (сравнение по алфавиту). Для сравнения по длине строки используется функция len().

Получить доступ к отдельному символу строки можно, указав его номер в квадратных скобках после имени переменной (строки).

Например, s[i], где i – номер символа строки s (индексация начинается с 0).

Пример 9. Сравнить содержимое двух строк по их длине:

```
s1 = input("Введите 1 строку: ")
print()
s2 = input("Введите 2 строку: ")
print()
if len(s1) > len(s2):
print("1 Строка длиннее")
elif len(s2) > len(s1):
print("2 Строка длиннее")
else:
print("Строки равные")
```

8.3 Методы обработки строк

Наиболее частые методы обработки строк в Python:

Таблица 8.1

Метод	Описание
s.upper()	Преобразует строку в верхний регистр
s.lower()	Преобразует строку в нижний регистр
s.capitalize()	Первая буква заглавная, остальные строчные
s.title()	Первая буква каждого слова заглавная
s.strip()	Удаляет пробелы в начале и конце строки

s.split()	Разбивает строку на список слов
s.join()	Объединяет элементы списка в строку
s.replace(old, new)	Заменяет подстроку old на new
s.find(sub)	Находит первое вхождение подстроки
s.count(sub)	Подсчитывает количество вхождений подстроки
len(s)	Возвращает длину строки

Пример 10. Вывести прописную и строчную версии строки:

```
s1 = input("Введите строку: ")
print()
s11 = s1.lower()
s12 = s1.upper()
print(f"Строчная версия строки: {s11}")
print()
print(f"Прописная версия строки: {s12}")
```

Пример 11. Вывести длину строки и ее копию.

```
s1 = "В программировании используется язык Python"
print()
print(f"s1: {s1}")
s2 = s1  # В Python строки неизменяемы, поэтому простое
присваивание создает копию
print()
print(f"s2: {s2}")
print()
print(f"Длина строки s1: {len(s1)}")
```

8.4. Строковые функции и процедуры

В Python строки являются объектами типа str, которые поддерживают Unicode кодировку. Символы можно задавать:

- символом, заключенным в одинарные или двойные кавычки;
- escape-последовательностью;
- Unicode-последовательностью, задающей Unicode код символа.

Примеры объявления символьных переменных и работа с ними:

```
ch1 = 'A'
ch2 = '\x59'  # буква Y Escape-последовательность \x59
ch3 = '\u0058'  # буква X кодировка Юникода \u0058
ch = ch1  # ch объявляется в объектном стиле
```

```
code = ord(ch)  # преобразование символьного типа в код
(int)
ch1 = chr(code + 1)  # формирование кода буквы B
# преобразование символьного типа в строку
s = ch1 + ch2 + ch3
print(f"s = {s}, ch = {ch}, code = {code}")
```

Строку можно задавать из:

- символа, повторенного заданное число раз;

- списка символов;

- части списка символов.

Примеры объявления строковых переменных и работа с ними:

```
world = "Мир"  # объявление переменной world
sssss = 's' * 5  # строка с буквой s повторяющейся 5 раз
yes = list("Yes")  # список символов для строки Yes
stryes = ''.join(yes)  # объявление строкового класса
stryes
strye = ''.join(yes[0:2])  # объявление строкового
класса для символьного
# списка yes с начальной позицией знака = 0 и длиной =
2 символа
print(f"world = {world}, sssss = {sssss}, stryes =
{stryes}, strye = {strye}")
```

8.5. Дополнительные возможности работы со строками в Python

Форматирование строк:

```
# f-строки (рекомендуемый способ)
name = "Иван"
age = 25
print(f"Меня зовут {name}, мне {age} лет")
# Метод format()
print("Меня зовут {}, мне {} лет".format(name, age))
# % форматирование
print("Меня зовут %s, мне %d лет" % (name, age))
```

Многострочные строки:

```
text = """Это многострочная
строка в Python
с переносами строк"""
```

Проверка содержимого строки:

```
s = "Hello World"
print(s.startswith("Hello"))  # True
print(s.endswith("World"))    # True
print(s.isdigit())            # False
print(s.isalpha())            # False (содержит пробел)
print("123".isdigit())        # True
```

Работа с индексами и срезами:

```
s = "Python"
print(s[0])      # P
print(s[-1])     # n
print(s[1:4])    # yth
print(s[:3])     # Pyt
print(s[3:])     # hon
print(s[::-1])   # nohtyP (обратная строка)
```

Краткие итоги. В лекции были рассмотрены определения символа и строки в Python. Изучены методы обработки строк. Задание символов и строк. Даны примеры объявления строковых переменных и символов. Рассмотрены дополнительные возможности Python для работы со строками.

Контрольные вопросы:

1. Какие переменные используются для хранения и обработки отдельных символов в Python?
2. Что относится к символам?
3. Как объявляются символы, приведите примеры.
4. Дайте определение строки в Python.
5. Как объявляются строковые данные?
6. Чем отличается переменная с инициализацией, от переменной без инициализации?
7. Назовите часто встречающиеся методы обработки строк в Python.
8. Как работает форматирование строк в Python?
9. Что такое f-строки?
10. Как получить доступ к отдельному символу строки

Лекция 9

Классы, объекты и методы. Функции в Python

Цель лекции: изучить классы, объекты и методы, их характеристики, определение, свойства и как они представляются. Рассмотреть функции их преимущества, дать определение.

9.1 Введение в классы, объекты и методы

Программирование в Python построено на классах. У класса две различные роли: модуля и типа данных.

Класс – это некий шаблон (модуль), который определяет форму объекта. Или множество объектов, связанных общностью структуры и поведения. Класс определяет, как должен быть построен объект. Однотипные объекты могут объединяться в классы (группы).

Конкретный объект, имеющий структуру этого класса, называется экземпляром класса.

Например, объект кнопка button1 – это экземпляр класса кнопок Button. Сам класс определяется общими свойствами, которые имеют все экземпляры этого класса.

Модульность построения – основное свойство программных систем. Система, построенная по модульному принципу, состоит из классов, являющихся основным видов модуля. Можно построить монолитную систему, состоящую из одного модуля, решающая ту же задачу, что и система, состоящая из нескольких простых модулей. Большую систему, создаваемую коллективом разработчиков, без разделения системы на модули построить не удается. Поэтому

модульность построения – основное средство борьбы со сложностью системы.

Класс – это особый тип записи, имеющий в своем составе атрибуты (поля) и методы обработки. Каждый модуль имеет содержательную начинку. Класс становится модулем и имеет определенное назначение. В основе класса лежит абстрактный тип данных.

Атрибуты класса служат для хранения информации об объекте. В Python атрибуты могут быть как обычными переменными, так и свойствами (properties), которые обеспечивают контролируемый доступ к данным.

Методами называются функции, предназначенные для обработки внутренних данных объекта данного класса (атрибутов). Объект может обладать набором заранее встроенных методов обработки, созданных пользователем, которые выполняются при наступлении заранее определенных событий, например: нажатие кнопки мыши, определенной клавиши, выбор пункта меню и т.п.

Свойства – характеристики объекта, его параметры. Объект представляется с помощью присущих ему свойств.

Например:

```
ОБЪЕКТ_1 (свойство-1, свойство-2, . . . свойство-k).
```

Свойства объектов различных классов могут пересекаться. Например:

```
ОБЪЕКТ_A (. . свойство-n, свойство-m, . . . свойство-k).
ОБЪЕКТ_B (свойство-1,. . . свойство-n, . . . свойство-k).
```

9.2 Синтаксис класса

```
class ИмяКласса:
def __init__(self, параметры):  # Конструктор
# объявление атрибутов экземпляра
self.атрибут1 = значение1
self.атрибут2 = значение2
# ... ... ... ...
self.атрибутN = значениеN
def метод1(self, параметры):  # объявление методов
# тело метода1
pass
def метод2(self, параметры):
# тело метода2
pass
# ... ... ... ...
def методN(self, параметры):
# тело методаN
pass
```

В Python нет явных спецификаторов доступа как в C#. По соглашению:

- атрибуты и методы, начинающиеся с одного подчеркивания (_), считаются внутренними

- атрибуты и методы, начинающиеся с двух подчеркиваний (__), считаются приватными

- все остальные атрибуты и методы являются публичными

1. Правильно определенный класс должен содержать логически связанные данные.
2. Все переменные, объявленные на уровне класса, являются атрибутами класса.
3. Атрибуты задают представления абстракции данных, которую реализует класс.
4. Атрибуты характеризуют свойства объектов класса.
5. Описаниями класса являются объявления атрибутов и методов.

Метод представляет собой функцию, являющуюся элементом класса. Методы выполняют действия над объектами класса. Все объекты одного класса имеют один и тот же набор методов.

9.3. Основные понятия функции

Функция – логически законченная часть программы, которую по имени можно вызывать в разные места программы неограниченное число раз.

Она решает часть общей задачи. У нее есть имя, которое используется при вызове функции в основную программу.

В Python все подпрограммы являются функциями. Функция может:

- возвращать значение с помощью оператора return
- не возвращать значение (возвращает None)
- возвращать несколько значений (в виде кортежа)

Функция отличается от процедуры двумя особенностями:

- всегда может вычислять некоторое значение, возвращаемое в качестве результата функции,
- функция может вызываться в выражениях.

В Python функция может быть реализована как:

- функция, возвращающая значение
- функция, не возвращающая значение (аналог процедуры)

Программа без структурных элементов называется монолитной. Минимальный элемент такой программы – оператор (инструкция). Она сложна в разработке, отладке и сопровождении.

Структурированная программа называется модульной. Она содержит более крупные компоненты – функции. Модульная программа проще создается, более понятна.

Программа на языке Python строится из модулей, роль которых играют классы и функции, но каждый из этих модулей имеет содержательную начинку.

Функции могут быть связаны с классом (методы), они обеспечивают требуемую функциональность класса и называются методами класса. Также функции могут существовать независимо от класса.

Синтаксис объявления функции позволяет однозначно определить, что является функцией.

Работа с функцией включает 2 этапа:

- описание функции;
- вызов функции.

Преимущества функции:

1. в программе нет дублирования кода;
2. повышается надежность программы;
3. улучшается «читаемость» программы;
4. облегчается процесс отладки.

Различают функции:

- без параметров (),
- с параметрами (params).

9.4 Сравнение вариантов

Рассмотрим 2 варианта решения одной и той же задачи:

Вывод текстовых блоков со вставкой стандартного разделителя из трех строк из набора символов " * " в конце.

1-й вариант (Монолитная программа)

```
def main():
print("Текст и разделители")
print()
print("Текст 1")  # Вывод 1 текста
for i in range(3):
for j in range(20):
print("*", end="")
print()  # Вывод строк разделителя
print()
print("Текст 2")  # Вывод 2 текста
for i in range(3):
for j in range(20):
print("*", end="")
print()  # Вывод строк разделителя
print()
print("Текст 3")  # Вывод 3 текста
for i in range(3):
for j in range(20):
print("*", end="")
print()  # Вывод строк разделителя
input()
if __name__ == "__main__":
main()
```

2-й вариант (Модульная программа)

```
def draw_str():  # Функция "Рисовать строки" без
параметров
for i in range(3):
for j in range(20):
print("*", end="")
print()  # Вывод строк разделителя
def main():
print("Текст и разделители")
print()
print("Текст 1")  # Вывод 1 текста
draw_str()  # Вызов функции "Рисовать строки"
print()
print("Текст 2")  # Вывод 2 текста
```

```
draw_str()  # Вызов функции "Рисовать строки"
print()
print("Текст 3")  # Вывод 3 текста
draw_str()  # Вызов функции "Рисовать строки"
input()
if __name__ == "__main__":
main()
```

9.5 Описание функций

Синтаксически в описании функции различают две части:

- Описание заголовка
- Описание тела функции.

Синтаксис заголовка функции:

```
def имя_функции([список параметров]):
имя_функции ([список параметров]) составляют сигнатуру функции
– обязательная часть в заголовке функции.
```

Примеры описания функций:

```
def A():  # функция без параметров
pass
def B():  # функция, возвращающая целое значение
return 0
def C(x, y):  # функция с параметрами
return x + y
def _private_method():  # внутренний метод (начинается с _)
pass
def __private_method():  # приватный метод (начинается с __)
pass
```

9.6 Аргументы и параметры

Параметры – данные, с которыми работает функция. Это внутренние данные для функции и перечисляются в ее заголовке. Список может содержать фиксированное число аргументов,

разделяемых запятой – для функции с параметрами или быть пустым – для функции без параметров.

Синтаксис объявления параметров:

```
def функция(параметр1, параметр2, *args, **kwargs):
pass
```

Например: (p1, p2, *args, **kwargs)

Параметры связаны с аргументами.

Аргументы – данные, передаваемые в функцию или возвращаемые из нее. Это внешние для функции данные, с которыми имеет дело вызывающая часть программы. В функции им соответствуют параметры. Указываются в списке аргументов при обращении к функции и являются фактическими параметрами.

Синонимы:

Параметры = формальные параметры, условно P.

Аргументы = фактические аргументы, условно A.

Можно для фиксированного числа формальных параметров передать функции произвольное число фактических аргументов. Для этого в списке формальных параметров необходимо задать *args для позиционных аргументов и **kwargs для именованных аргументов.

Для аргументов и параметров надо соблюдать:

- одинаковое количество (An=Pn) для обязательных параметров,
- совместимость типов (типы одинаковы или неявно преобразованные).

Все формальные параметры разделяются на группы:

- *позиционные параметры* – передают информацию функции и их значения в теле функции только читаются;

- *именованные параметры* – имеют значения по умолчанию, могут быть переданы при вызове или использованы по умолчанию;

**args* – собирает дополнительные позиционные аргументы в кортеж;

***kwargs* – собирает дополнительные именованные аргументы в словарь.

Например:

```
# Функция с различными типами параметров
def example_function(p1, p2=10, *args, **kwargs):
print(f"p1: {p1}")
print(f"p2: {p2}")
print(f"args: {args}")
print(f"kwargs: {kwargs}")
# Вызов функции
example_function(1, 2, 3, 4, 5, name="John", age=25)
```

Структура функции идентична структуре программы.

Имена, объявленные в главной программе, являются *глобальными*. Они доступны во всех внутренних точках, в том числе и в функциях. Имена, объявленные в функции, являются *локальными*. Они доступны во всех внутренних точках функции.

Локальное имя во внешней программе недоступно. Память под них выделяется автоматически в момент вызова функции. Если используются одинаковые локальное и глобальное имя, то внутри функции локальное имя блокирует глобальное.

9.7 Тело функции

Синтаксически тело функции является блоком, который представляет собой последовательность инструкций и описание переменных с отступом.

В теле функции в блоке может быть оператор, возвращающий значение функции в формате return выражение

Переменные, описанные в блоке, считаются локальными в этом блоке.

Вызов функции. Синтаксис.

Функция может быть вызвана в выражении или как оператор тела блока. Сам вызов функции имеет один и тот же синтаксис:

```
имя_функции([список фактических аргументов])
```

В момент вызова функции происходит:

1. вычисление фактических аргументов, которые являются выражением,
2. в точке вызова создается блок, в котором происходит замена имен параметров фактическими аргументами.

Функция

Пример 1. Написать функцию, проверяющую, является ли целое число, введенное с клавиатуры четным. Программа должна использовать эту функцию, проверять является ли число четным или нет, и выводить соответствующее сообщение.

```
def chet(n):  # Функция "Четность"
if n % 2 == 0:
return True
else:
return False
def main():
print("Определение четности аргумента")
print()
while True:
m = int(input("m >> "))
if chet(m):
print("Число четное")
else:
print("Число нечетное")
print()
if m == 0:
break
input()
if __name__ == "__main__":
```

```
main()
```

Функция (аналог процедуры)

Пример 2. Вывести таблицу квадратных корней.

Для оформления таблицы использовать функцию Line.

```
import math
def line(n, c):  # Функция "Рисовать линию" с
параметрами n и c
for k in range(n):
print(c, end="")
print()  # Вывод строки с символами c
def main():
m = 18
print("Таблица квадратных корней")
print()
line(m, '=')  # Вызов функции "Рисовать линию"
print(" Число | Корень")
line(m, '=')  # Вызов функции "Рисовать линию"
for i in range(10):
q = math.sqrt(i)
print(f"   {i}   |   {q:3.3f}")
line(m, '-')  # Вызов функции "Рисовать линию"
input()
if __name__ == "__main__":
main()
```

9.8 Классы в Python - примеры

Пример 3. Простой класс для работы с прямоугольником:

```
class Rectangle:
def __init__(self, width, height):
self.width = width
self.height = height
def area(self):
return self.width * self.height
def perimeter(self):
return 2 * (self.width + self.height)
def __str__(self):
return f"Прямоугольник {self.width}x{self.height}"
# Использование класса
rect = Rectangle(5, 3)
print(rect)
print(f"Площадь: {rect.area()}")
print(f"Периметр: {rect.perimeter()}")
```

Пример 4. Класс с приватными атрибутами и свойствами:

```
class BankAccount:
def __init__(self, initial_balance=0):
self.__balance = initial_balance  # приватный атрибут
@property
def balance(self):  # свойство для чтения баланса
return self.__balance
def deposit(self, amount):
if amount > 0:
self.__balance += amount
return True
return False
def withdraw(self, amount):
if 0 < amount <= self.__balance:
self.__balance -= amount
return True
return False
# Использование класса
account = BankAccount(1000)
print(f"Баланс: {account.balance}")
account.deposit(500)
print(f"Баланс после пополнения: {account.balance}")
account.withdraw(200)
print(f"Баланс после снятия: {account.balance}")
```

Краткие итоги. В лекции были даны определения, что такое классы, объекты, методы в Python. Представлен синтаксис класса, методы представления функций. Изучены функции, относящиеся к подпрограмме, дано их описание. Рассмотрены особенности Python в работе с классами и функциями.

Контрольные вопросы:

1. Дайте определение понятию «класс» в Python.
2. Что лежит в основе класса?
3. Каковы свойства и характеристики объекта?
4. Как работают модификаторы доступа в Python?
5. Дайте определение понятию «функция» в Python.

6. Назовите преимущества функции.
7. Сравните монолитную и модульную программы, назовите их различия.
8. Опишите методы функций.
9. Дайте определение что такое «параметры» и «аргументы».
10. Каковы области действия имен в Python?
11. Опишите процедуру вызова функции.
12. Приведите пример работы функции.
13. Приведите пример работы класса.
14. Что такое self в Python?
15. Как работают свойства (properties) в Python?

Лекция 10
Исключения в Python

Цель лекции: разобраться что такое исключения, как они проявляются в работе. Изучить обработку исключения, какие обработчики используются.

Нарушение в работе программы (не корректное действие) называется исключением. Исключение останавливает текущий поток программы и если никакие меры не предпринимаются, программа просто прекращает выполнение. Причиной исключений могут быть ошибки в программе (например, деление числа на ноль) или неожиданный ввод (например, выбор несуществующего файла, или вместо цифры для выполнения арифметических действий – буква). Задачей пользователя является предоставление программе возможности устранить проблемы, не приводя к сбою.

Обработку исключений может выполнять:

- системный обработчик исключений или
- написанный пользователем программный код, который выводит сообщение.

Для обработки исключений используются глобальные и локальные обработчики исключений.

Глобальные обработчики – стандартные предусмотрены операционной системой и вызываются автоматически, если отладчик подключен. Обработчик завершает выполнение программы, выдавая информацию о возникновении исключения. Он не всегда понятен пользователю, не говоря уже о досрочном завершении программы.

Локальные обработчики создает сам пользователь в виде сообщений. Программист может перехватить исключительные ситуации и сделать их обработку, после чего продолжится нормальный ход вычислений приложения.

Обработка локальных исключений в Python реализована с помощью ключевых слов — try, except, else и finally — с помощью которых программа обнаруживает исключения, устраняет их и продолжает выполнение. Они способствуют повышению надежности приложений.

10.1 Виды исключений:

Исключения имеют типы, являющиеся производными от BaseException. Некоторые исключения, генерируемые при выполнении:

Таблица 10.1 – Типы исключений и их описание в Python

Тип исключения	**Описание**
ZeroDivisionError	Деление на ноль
IndexError	Индекс списка вне диапазона
KeyError	Ключ не найден в словаре
ValueError	Неправильное значение
TypeError	Неправильный тип данных
FileNotFoundError	Файл не найден
NameError	Переменная не определена
AttributeError	Атрибут не найден
ImportError	Ошибка импорта модуля
KeyboardInterrupt	Прерывание с клавиатуры (Ctrl+C)
SystemExit	Выход из программы

10.2 Типы блоков работающих с исключением

Существуют несколько типов блоков:

I. try … except # попробовать исключить

II. try … except … else # попробовать исключить с альтернативой

III. try … except … finally # попробовать исключить в финал

IV. try … except … else … finally # полная форма

I. Try, Except

Ключевые слова try и except используются вместе как составной оператор. Для таких блоков характерна вложенность. Если предполагается, что блок кода может вызвать исключение, используют ключевое слово try и except (тип_исключения), чтобы сохранить код, который будет выполнен при возникновении исключения.

Блок, которому предшествует ключевое слово try, называется охраняемым (контролируемым) блоком или try-блоком. Блок, которому предшествует конструкция except (тип_исключения), называется блоком перехватчиком исключения или except-блоком.

При отсутствии блоков try и except произойдет сбой программы.

Синтаксис инструкции:

```
try:
# Блок кода, подлежащий проверки на наличие ошибок
except ТипИсключения1:
# Код обработчика исключения (ТипИсключения1)
except ТипИсключения2:
# Код обработчика исключения (ТипИсключения2)
except:
# Обработчик для всех остальных исключений
```

С try-блоком может быть связана не одна, а несколько except-инструкций. Какая из них будет выполнена, определит тип исключения. Будет выполнена та except-инструкция, тип исключения которой совпадает с типом сгенерированного исключения (а все остальные будут проигнорированы).

Если исключение не сгенерируется, то try-блок завершается нормально, и все его except-инструкции игнорируются. Выполнение

программы продолжается с первой инструкции, которая стоит после последней инструкции except. Таким образом, except-инструкция выполняется только в случае, если сгенерировано соответствующее исключение.

Пример 1, в котором в результате деления на ноль создается исключение, которое затем перехватывается. В программе генерируется исключение типа ZeroDivisionError – деление на 0

```
x = 0
y = 0  # знаменатель равен 0
try:
x = 10 / y  # Проверяемая инструкция, возможно деление на 0
print(f"Ответ x= {x}")
except ZeroDivisionError:  # Обработчик исключения
print("Попытка деления на 0.")
```

Блоки try включают операторы программы, которые могут вызвать исключительную ситуацию, например y=0.

Теперь заменим y=0 на y=2:

```
x = 0
y = 2  # знаменатель равен, например 2 (отличен от 0)
try:
x = 10 / y  # Проверяемая инструкция, возможно деление на 0
print(f"Ответ x= {x}")
except ZeroDivisionError:  # Обработчик исключения
print("Попытка деления на 0.")
```

Пример 2, в котором происходит индексация списка и попытка индексировать список за пределами его границ, вызывает ошибку нарушения диапазона.

```
def main():
N = [0] * 4  # список для 4-х элементов
try:
print("Перед генерированием исключения.")
# связанное с попаданием индекса вне диапазона
for i in range(10):  # в цикле индексируется список от 0 до 9
N[i] = i
print(f"N[{i}]:{N[i]}")
```

```
print("Этот текст не отображается")
except IndexError:
# Перехватываем исключение.
print("Индекс вне диапазона")
print("После except-инструкции")
input()
if __name__ == "__main__":
main()
```

В программе намеренно генерируется исключение типа IndexError, а затем это исключение перехватывается.

В программе объявляется список для 4-х элементов, а в цикле делается попытка индексировать этот список от 0 до 9. Как только значение индекса устанавливается равным четырем, генерируется исключение типа IndexError.

Ключевые аспекты обработки исключений:

1. проверяемый код содержится внутри try-блока,

2. при возникновении исключения выполнение try-блока прекращается, а само исключение перехватывается except инструкцией,

3. except инструкция не вызывается, а ей передается управление программой.

4. после выполнения except инструкции программа продолжится со следующей инструкции.

Если try-блоком исключение не сгенерировано, ни одна из except инструкций не выполняется и управление программой будет передано инструкции, следующей за except инструкций.

Если заменить в цикле for i in range(10) верхний индекс на for i in range(len(N)), то границы индексирования списка не нарушаются. Поэтому исключение не генерируется и except-блок не выполняется.

Если не создать локального обработчика исключения, то Python система динамического управления перехватит исключение, сообщит об ошибке и завершит программу.

С try-блоком можно связать несколько except инструкций. Все except инструкции должны перехватывать исключения различного типа.

Пример 3. Программа перехватывает как ошибку нарушения границ списка, так и ошибку деления на нуль.

```
def main():
N = [4, 8, 16, 32, 64, 128, 256, 512]  # список из 8
элементов
D = [2, 0, 4, 4, 0, 8]  # список из 6 элементов
for i in range(len(N)):  # в цикле по всем элементам
try:
print(f"{N[i]} / {D[i]} = {N[i]/D[i]}")
except ZeroDivisionError:
# Перехватываем исключение.
print("На нуль делить нельзя!")
except IndexError:
# Перехватываем исключение.
print("Нет соответствующего элемента.")
input()
if __name__ == "__main__":
main()
```

Каждая except инструкция реагирует только на собственный тип исключения.

II. Try, Except, Else

Блок else выполняется только если в try-блоке не возникло исключений.

```
try:
# Код, который может вызвать исключение
result = 10 / 2
except ZeroDivisionError:
print("Деление на ноль!")
else:
print(f"Результат: {result}")
```

III. Try, Except, Finally

Иногда возникает потребность определить программный блок, который должен выполняться по выходу из try/except блока. Исключение может вызвать ошибку, которая является причиной преждевременного возврата из текущего метода. Удобный путь выхода из этого – блок finally.

Код, содержащийся в блоке finally, выполняется всегда, вне зависимости от возникновения исключения. Чтобы гарантировать возвращение ресурсов, например, убедиться, что файл закрыт, или освободить память от локальных переменных.

Синтаксис инструкции:

```
try:
# Блок кода, предназначенный для обработки ошибок
except ТипИсключения1:
# Обработчик для исключения (ТипИсключения1).
except ТипИсключения2:
# Обработчик для исключения (ТипИсключения2).
finally:
# Код завершения обработки исключений.
```

Не зависимо от итога выполнения try/except блоков, блок finally выполняется обязательно.

Пример 4. Используя закон Ома рассчитать величину тока. Проект консольное приложение.

```
def calculate_current():
try:
U = int(input("Введите напряжение (U): "))  #
преобразование строкового
R = int(input("Введите сопротивление (R): "))  #
отображения в целое
I = U / R
print(f"Ток I = {I:.2f} A")  # преобразование числа в
строку
except ValueError:
print("Нельзя вводить буквы и символы")
except ZeroDivisionError:
print("Сопротивление не может быть равно нулю")
finally:
```

```
print("Расчет завершен")
if __name__ == "__main__":
calculate_current()
```

10.3 Создание собственных исключений

В Python можно создавать собственные классы исключений, наследуя их от базового класса Exception.

```
class CustomError(Exception):
"""Пользовательское исключение"""
pass
def check_positive_number(value):
if value < 0:
raise CustomError("Число должно быть положительным")
return value
try:
result = check_positive_number(-5)
except CustomError as e:
print(f"Ошибка: {e}")
```

10.4 Обработка исключений с получением информации

Можно получить дополнительную информацию об исключении:

```
try:
x = 10 / 0
except ZeroDivisionError as e:
print(f"Ошибка: {e}")
print(f"Тип ошибки: {type(e).__name__}")
```

10.5 Форматы ввода/вывода

При вводе и выводе информации в языке Python используются функции преобразования:

Таблица 10.2 – Функции преобразования и их значения в Python

Функция	Описание	Пример
int()	Преобразование в целое число	int("123") → 123
float()	Преобразование в вещественное число	float("3.14") → 3.14
str()	Преобразование в строку	str(123) → "123"

bool()	Преобразование в логическое значение	bool(1) → True
list()	Преобразование в список	list("abc") → ['a', 'b', 'c']
tuple()	Преобразование в кортеж	tuple([1,2,3]) → (1, 2, 3)
dict()	Преобразование в словарь	dict([('a',1)]) → {'a': 1}

Например, для консольного приложения:

```
x = float(input())  # вещественный тип
d = int(input())  # целый тип
z = int(input())  # целый тип
ch = input()[0]  # символьный тип (первый символ)
print(f"x={x:3.2f} y={y:5.2f}")  # форматный вывод
переменных x и y,
# где 3 и 5 – количество выводимых символов,
# 2 – два символа после запятой
```

10.6 Обработка исключений при работе с файлами

Пример безопасной работы с файлами:

```
def read_file_safely(filename):
try:
with open(filename, 'r', encoding='utf-8') as file:
content = file.read()
print("Файл успешно прочитан")
return content
except FileNotFoundError:
print(f"Файл {filename} не найден")
except PermissionError:
print(f"Нет прав доступа к файлу {filename}")
except Exception as e:
print(f"Произошла ошибка: {e}")
finally:
print("Обработка файла завершена")
# Использование
content = read_file_safely("example.txt")
```

10.7 Контекстные менеджеры (with)

Python предоставляет удобный способ работы с ресурсами через контекстные менеджеры:

```
# Автоматическое закрытие файла
with open("file.txt", "r") as f:
```

```
content = f.read()
# Файл автоматически закроется здесь
# Создание собственного контекстного менеджера
class MyContextManager:
def __enter__(self):
print("Вход в контекст")
return self
def __exit__(self, exc_type, exc_val, exc_tb):
print("Выход из контекста")
if exc_type:
print(f"Произошло исключение: {exc_val}")
return False  # Не подавлять исключение
# Использование
with MyContextManager() as cm:
print("Работа в контексте")
# raise ValueError("Тестовая ошибка")
```

Краткие итоги. В лекции были рассмотрены исключения, и работа с ними, посредством обработчиков. Приведены примеры работы блоков работающих с исключениями. Рассмотрены особенности Python в обработке исключений.

Контрольные вопросы.

1. Дайте определение понятию «исключение» в Python.
2. Как можно выполнить обработку исключений?
3. Перечислите обработчики исключений и их принцип работы.
4. Виды исключений, типы и описания в Python.
5. Блоки, работающие с исключениями, коротко охарактеризуйте их.
6. Ключевые слова try и except, как они используются в блоке исключения.
7. Приведите пример использования try и except.
8. Ключевые аспекты обработки исключения при использования try и except.
9. Программный блок использующий try, except, finally.

10.Приведите пример использования try, except, finally.

11.Функции преобразования и их значения в Python.

12.Приведите примеры формата ввода и вывода.

13.Как создать собственное исключение в Python?

14.Что такое контекстные менеджеры?

Лекция 11

Приложение под ОС Windows в Python

Цель лекции: изучить приложение ОС Windows, его основные характеристики и принцип работы с ним. Разобрать назначение окон присутствующих в GUI приложениях на Python.

11.1. Основные характеристики приложения GUI в Python

При создании GUI приложений в Python предоставляется возможность использовать различные библиотеки для создания графического интерфейса под операционную систему Windows:

- tkinter (встроенная библиотека)
- PyQt5/PyQt6
- PySide2/PySide6
- wxPython
- Kivy

tkinter является стандартной библиотекой Python и не требует дополнительной установки. Она позволяет создавать приложения с графическим интерфейсом, отличающиеся небольшими размерами исполняемого файла.

Основные компоненты tkinter:

- Главное окно (Tk)
- Виджеты (компоненты интерфейса)
- События и обработчики
- Менеджеры размещения (pack, grid, place)

11.2 Основные виджеты tkinter

Окно содержит:

- Строку заголовка, которая отображает имя приложения;
- Строку меню с набором команд для разработки, тестирования приложений;
- Панель инструментов с кнопками, соответствующие основным командам;
- Область содержимого – средство отображения приложения;
- Виджеты (компоненты) для создания графического интерфейса.

Элементы подразделяются на:

- Визуальные (кнопки, меню, поля редактирования, переключатели, списки…),
- Логические (окна диалога, отчеты, часы…), отображающиеся в процессе запуска приложения.

11.3 Создание простого GUI приложения

Простейшее приложение представляет собой заготовку, обеспечивающую все необходимое для приложения. Это главное окно, для которого уже созданы базовые настройки.

Форма отображается при первом запуске и содержит основные элементы окна Windows:

- заголовок приложения,
- кнопки минимизации, максимизации и закрытия окна, изменения размеров окна.

При создании приложений на форму помещаются виджеты, для которых устанавливаются свойства и создаются обработчики событий.

Пример простейшего приложения:

```
import tkinter as tk
from tkinter import ttk
class SimpleApp:
def __init__(self, root):
self.root = root
self.root.title("Простое приложение")
self.root.geometry("400x300")
# Создание виджетов
self.create_widgets()
def create_widgets(self):
# Метка
self.label = tk.Label(self.root, text="Добро
пожаловать!")
self.label.pack(pady=10)
# Кнопка
self.button = tk.Button(self.root, text="Нажми меня",
command=self.on_button_click)
self.button.pack(pady=5)
# Поле ввода
self.entry = tk.Entry(self.root, width=30)
self.entry.pack(pady=5)
# Текстовое поле
self.text = tk.Text(self.root, height=5, width=40)
self.text.pack(pady=5)
def on_button_click(self):
text = self.entry.get()
self.text.insert(tk.END, f"Вы ввели: {text}\n")
self.entry.delete(0, tk.END)
# Создание и запуск приложения
if __name__ == "__main__":
root = tk.Tk()
app = SimpleApp(root)
root.mainloop()
```

11.4 Свойства виджетов

Свойства виджета это атрибуты, определяющие способ отображения и функционирования компонента.

Типы свойств:

- простые – это те, значения которых являются числами или строками. Например, text, width, height, bg, fg;

- перечисляемые – это те, которые могут принимать значения из предложенного набора (списка). Например, relief, cursor, state;
- вложенные – это те, которые поддерживают вложенные значения (или объекты). Например, font, padx, pady.

Изменять свойства виджетам можно двумя способами:

- При создании виджета (в конструкторе);
- Программно – путем создания соответствующих методов с соответствующими строками кода.

При выполнении приложения свойства компонентов можно изменять с помощью операторов присваивания.

Например:

```
# изменение программно надписи кнопки button1:
self.button.config(text="Решить")
# изменение размеров текстового поля:
self.text.config(height=10, width=50)
# изменение текста и цвета шрифта в метке:
self.label.config(text="Студент", fg="red")
```

11.5 События и обработчики

Событие – это то, что происходит во время работы программы. В Python каждому событию присваивается имя.

Реакция окна на разного рода действия пользователя определяет функциональность приложения.

Обработчик событий – определяет действие, которое можно поручить выделенному компоненту. Каждый компонент имеет набор стандартных обработчиков событий.

Основные события:

- Button: <Button-1> (щелчок левой кнопкой мыши)
- Entry: <KeyPress>, <FocusIn>, <FocusOut>

- Text: <KeyPress>, <Button-1>

- Window: <Configure>, <Destroy>

Пример создания обработчика событий:

```
import tkinter as tk
from tkinter import messagebox
class EventApp:
def __init__(self, root):
self.root = root
self.root.title("Приложение с событиями")
self.root.geometry("400x300")
self.create_widgets()
self.bind_events()
def create_widgets(self):
self.label = tk.Label(self.root, text="Нажмите кнопку
или введите текст")
self.label.pack(pady=10)
self.button = tk.Button(self.root, text="Кнопка")
self.button.pack(pady=5)
self.entry = tk.Entry(self.root, width=30)
self.entry.pack(pady=5)
self.text = tk.Text(self.root, height=5, width=40)
self.text.pack(pady=5)
def bind_events(self):
# Обработчик клика по кнопке
self.button.bind('<Button-1>', self.on_button_click)
# Обработчик нажатия клавиши в поле ввода
self.entry.bind('<KeyPress>', self.on_key_press)
# Обработчик изменения размера окна
self.root.bind('<Configure>', self.on_window_resize)
def on_button_click(self, event):
self.label.config(text="Кнопка нажата!")
self.text.insert(tk.END, "Кнопка была нажата\n")
def on_key_press(self, event):
self.label.config(text=f"Нажата клавиша:
{event.char}")
def on_window_resize(self, event):
if event.widget == self.root:
self.text.insert(tk.END, f"Окно изменено:
{event.width}x{event.height}\n")
if __name__ == "__main__":
root = tk.Tk()
app = EventApp(root)
root.mainloop()
```

11.6 Менеджеры размещения

В tkinter существует три основных менеджера размещения:

1. pack() - размещение в блоках
2. grid() - размещение в сетке
3. place() - абсолютное размещение

Пример использования grid():

```
import tkinter as tk
from tkinter import ttk
class GridApp:
def __init__(self, root):
self.root = root
self.root.title("Приложение с сеткой")
self.root.geometry("400x300")
self.create_widgets()
def create_widgets(self):
# Создание виджетов с размещением в сетке
tk.Label(self.root, text="Имя:").grid(row=0, column=0,
sticky="w", padx=5, pady=5)
self.name_entry = tk.Entry(self.root, width=20)
self.name_entry.grid(row=0, column=1, padx=5, pady=5)
tk.Label(self.root, text="Возраст:").grid(row=1,
column=0, sticky="w", padx=5, pady=5)
self.age_entry = tk.Entry(self.root, width=20)
self.age_entry.grid(row=1, column=1, padx=5, pady=5)
tk.Button(self.root, text="Сохранить",
command=self.save_data).grid(row=2, column=0, padx=5,
pady=5)
tk.Button(self.root, text="Очистить",
command=self.clear_data).grid(row=2, column=1, padx=5,
pady=5)
self.result_text = tk.Text(self.root, height=10,
width=40)
self.result_text.grid(row=3, column=0, columnspan=2,
padx=5, pady=5)
def save_data(self):
name = self.name_entry.get()
age = self.age_entry.get()
self.result_text.insert(tk.END, f"Имя: {name},
Возраст: {age}\n")
def clear_data(self):
self.name_entry.delete(0, tk.END)
self.age_entry.delete(0, tk.END)
self.result_text.delete(1.0, tk.END)
```

```
if __name__ == "__main__":
root = tk.Tk()
app = GridApp(root)
root.mainloop()
```

11.7 Диалоговые окна

tkinter предоставляет стандартные диалоговые окна:

```
import tkinter as tk
from tkinter import messagebox, filedialog,
colorchooser
class DialogApp:
def __init__(self, root):
self.root = root
self.root.title("Приложение с диалогами")
self.root.geometry("400x300")
self.create_widgets()
def create_widgets(self):
tk.Button(self.root, text="Сообщение",
command=self.show_message).pack(pady=5)
tk.Button(self.root, text="Предупреждение",
command=self.show_warning).pack(pady=5)
tk.Button(self.root, text="Ошибка",
command=self.show_error).pack(pady=5)
tk.Button(self.root, text="Вопрос",
command=self.show_question).pack(pady=5)
tk.Button(self.root, text="Открыть файл",
command=self.open_file).pack(pady=5)
tk.Button(self.root, text="Сохранить файл",
command=self.save_file).pack(pady=5)
tk.Button(self.root, text="Выбрать цвет",
command=self.choose_color).pack(pady=5)
def show_message(self):
messagebox.showinfo("Информация", "Это информационное
сообщение")
def show_warning(self):
messagebox.showwarning("Предупреждение", "Это
предупреждение")
def show_error(self):
messagebox.showerror("Ошибка", "Это сообщение об
ошибке")
def show_question(self):
result = messagebox.askyesno("Вопрос", "Вы уверены?")
if result:
messagebox.showinfo("Ответ", "Вы ответили 'Да'")
else:
```

```
messagebox.showinfo("Ответ", "Вы ответили 'Нет'")
def open_file(self):
filename = filedialog.askopenfilename()
if filename:
messagebox.showinfo("Файл", f"Выбран файл:
{filename}")
def save_file(self):
filename = filedialog.asksaveasfilename()
if filename:
messagebox.showinfo("Файл", f"Файл будет сохранен как:
{filename}")
def choose_color(self):
color = colorchooser.askcolor()
if color[1]:
messagebox.showinfo("Цвет", f"Выбран цвет:
{color[1]}")
if __name__ == "__main__":
root = tk.Tk()
app = DialogApp(root)
root.mainloop()
```

11.8 Меню и панели инструментов

```
import tkinter as tk
from tkinter import ttk, messagebox
class MenuApp:
def __init__(self, root):
self.root = root
self.root.title("Приложение с меню")
self.root.geometry("500x400")
self.create_menu()
self.create_toolbar()
self.create_widgets()
def create_menu(self):
menubar = tk.Menu(self.root)
self.root.config(menu=menubar)
# Файл
file_menu = tk.Menu(menubar, tearoff=0)
menubar.add_cascade(label="Файл", menu=file_menu)
file_menu.add_command(label="Новый",
command=self.new_file)
file_menu.add_command(label="Открыть",
command=self.open_file)
file_menu.add_separator()
file_menu.add_command(label="Выход",
command=self.root.quit)
```

```
# Правка
edit_menu = tk.Menu(menubar, tearoff=0)
menubar.add_cascade(label="Правка", menu=edit_menu)
edit_menu.add_command(label="Копировать",
command=self.copy)
edit_menu.add_command(label="Вставить",
command=self.paste)
# Справка
help_menu = tk.Menu(menubar, tearoff=0)
menubar.add_cascade(label="Справка", menu=help_menu)
help_menu.add_command(label="О программе",
command=self.about)
def create_toolbar(self):
toolbar = ttk.Frame(self.root)
toolbar.pack(side=tk.TOP, fill=tk.X)
ttk.Button(toolbar, text="Новый",
command=self.new_file).pack(side=tk.LEFT, padx=2)
ttk.Button(toolbar, text="Открыть",
command=self.open_file).pack(side=tk.LEFT, padx=2)
ttk.Button(toolbar, text="Сохранить",
command=self.save_file).pack(side=tk.LEFT, padx=2)
ttk.Separator(toolbar,
orient=tk.VERTICAL).pack(side=tk.LEFT, fill=tk.Y,
padx=5)
ttk.Button(toolbar, text="Копировать",
command=self.copy).pack(side=tk.LEFT, padx=2)
ttk.Button(toolbar, text="Вставить",
command=self.paste).pack(side=tk.LEFT, padx=2)
def create_widgets(self):
self.text = tk.Text(self.root, wrap=tk.WORD)
self.text.pack(fill=tk.BOTH, expand=True, padx=5,
pady=5)
def new_file(self):
self.text.delete(1.0, tk.END)
messagebox.showinfo("Файл", "Создан новый файл")
def open_file(self):
messagebox.showinfo("Файл", "Открытие файла")
def save_file(self):
messagebox.showinfo("Файл", "Сохранение файла")
def copy(self):
messagebox.showinfo("Правка", "Копирование")
def paste(self):
messagebox.showinfo("Правка", "Вставка")
def about(self):
messagebox.showinfo("О программе", "Простое GUI
приложение на Python")
if __name__ == "__main__":
```

```
root = tk.Tk()
app = MenuApp(root)
root.mainloop()
```

11.9 Обработка ошибок в GUI приложениях

```
import tkinter as tk
from tkinter import messagebox
class ErrorHandlingApp:
def __init__(self, root):
self.root = root
self.root.title("Приложение с обработкой ошибок")
self.root.geometry("400x300")
self.create_widgets()
def create_widgets(self):
tk.Label(self.root, text="Введите два
числа:").pack(pady=5)
self.entry1 = tk.Entry(self.root, width=20)
self.entry1.pack(pady=5)
self.entry2 = tk.Entry(self.root, width=20)
self.entry2.pack(pady=5)
tk.Button(self.root, text="Сложить",
command=self.add_numbers).pack(pady=5)
tk.Button(self.root, text="Разделить",
command=self.divide_numbers).pack(pady=5)
self.result_label = tk.Label(self.root,
text="Результат появится здесь")
self.result_label.pack(pady=10)
def add_numbers(self):
try:
num1 = float(self.entry1.get())
num2 = float(self.entry2.get())
result = num1 + num2
self.result_label.config(text=f"Сумма: {result}")
except ValueError:
messagebox.showerror("Ошибка", "Пожалуйста, введите
корректные числа")
except Exception as e:
messagebox.showerror("Ошибка", f"Произошла ошибка:
{e}")
def divide_numbers(self):
try:
num1 = float(self.entry1.get())
num2 = float(self.entry2.get())
if num2 == 0:
raise ZeroDivisionError("Деление на ноль")
```

```
result = num1 / num2
self.result_label.config(text=f"Частное: {result}")
except ValueError:
messagebox.showerror("Ошибка", "Пожалуйста, введите
корректные числа")
except ZeroDivisionError:
messagebox.showerror("Ошибка", "Деление на ноль
невозможно")
except Exception as e:
messagebox.showerror("Ошибка", f"Произошла ошибка:
{e}")
if __name__ == "__main__":
root = tk.Tk()
app = ErrorHandlingApp(root)
root.mainloop()
```

11.10 Создание исполняемого файла

Для создания исполняемого файла из Python приложения можно использовать PyInstaller:

```
# Установка PyInstaller
# pip install pyinstaller
# Создание исполняемого файла
# pyinstaller --onefile --windowed your_app.py
# --onefile: создает один исполняемый файл
# --windowed: скрывает консольное окно для GUI приложений
```

Краткие итоги. В лекции были рассмотрены вопросы, относящиеся к созданию GUI приложений в Python. Даны основные характеристики tkinter. Описан процесс создания приложения с графическим интерфейсом. Рассмотрены основные виджеты, события и обработчики.

Контрольные вопросы

1. Перечислите основные библиотеки для создания GUI в Python.
2. Перечислите основные виджеты tkinter.
3. Что необходимо для размещения виджета в окне.

4. Что отражают свойства виджетов?
5. Дайте краткую характеристику tkinter.
6. Перечислите этапы разработки GUI приложения.
7. Что необходимо для создания интерфейса?
8. Как обеспечить функциональность приложения?
9. Перечислите типы свойств виджетов.
10. Как можно изменить свойства виджета?
11. Приведите примеры изменения свойств виджета.
12. Дайте определение понятию «Событие» в GUI.
13. Какие менеджеры размещения существуют в tkinter?
14. Как создать диалоговое окно в tkinter?
15. Как обработать ошибки в GUI приложении?

Лекция 12
Графика в Python

Цель лекции: ознакомиться с правилами работы с графикой в Python. Научиться загружать готовые картинки, рисовать комбинированные фигуры, менять цвет, шрифт.

Python позволяет:

- загрузить в приложение готовые картинки и фотографии,
- разработать программы, которые выводят графику на поверхность объекта.

В Python можно загружать файлы – рисунки, созданные в других программах, например Paint, Corel Draw и др.

12.1. Рисованные изображения

Рисованные изображения отображаются на форме при выполнении программы с помощью различных инструментов.

Изображение при этом представляет собой комбинацию простейших фигур – графических примитивов (точка, линия, круг или прямоугольник).

Каждая точка на форме имеет координаты X и Y. Текущая позиция при рисовании определяется горизонтальной (X) и вертикальной (Y) координатами, заданными в пикселях. Координата X возрастает при перемещении указателя слева на право, а координата Y – при перемещении его сверху вниз.

12.2 Библиотеки для работы с графикой в Python

В Python для работы с графикой используются различные библиотеки:

- **matplotlib** - для создания графиков и диаграмм
- **PIL (Pillow)** - для работы с изображениями
- **tkinter** - для простой графики в GUI приложениях
- **pygame** - для игровой графики
- **PIL + tkinter** - для отображения изображений в GUI

12.3 Работа с matplotlib

matplotlib - основная библиотека для создания графиков в Python.

```
import matplotlib.pyplot as plt
import numpy as np
# Создание простого графика
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
plt.figure(figsize=(8, 6))
plt.plot(x, y, 'b-', linewidth=2, label='sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.title('График функции sin(x)')
plt.grid(True)
plt.legend()
plt.show()
```

12.4 Рисование примитивов с помощью matplotlib

```
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.patches import Circle, Rectangle,
Polygon
# Создание фигуры и осей
fig, ax = plt.subplots(figsize=(10, 8))
# Рисование линий
ax.plot([0, 1, 2, 3], [0, 1, 0, 1], 'r-', linewidth=2,
label='Линия')
# Рисование прямоугольника
```

```
rect = Rectangle((0.5, 0.5), 1, 0.5, linewidth=2,
edgecolor='blue', facecolor='lightblue')
ax.add_patch(rect)
# Рисование круга
circle = Circle((2, 2), 0.5, linewidth=2,
edgecolor='green', facecolor='lightgreen')
ax.add_patch(circle)
# Рисование многоугольника
polygon = Polygon([(3, 0), (3.5, 1), (3, 2), (2.5,
1)], linewidth=2, edgecolor='purple',
facecolor='lightpink')
ax.add_patch(polygon)
# Настройка осей
ax.set_xlim(0, 4)
ax.set_ylim(0, 3)
ax.set_aspect('equal')
ax.grid(True, alpha=0.3)
ax.legend()
ax.set_title('Графические примитивы')
plt.show()
```

12.5 Работа с текстом и шрифтами

```
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
# Создание фигуры
fig, ax = plt.subplots(figsize=(10, 6))
# Различные стили текста
ax.text(0.1, 0.8, 'Обычный текст', fontsize=12,
transform=ax.transAxes)
ax.text(0.1, 0.7, 'Жирный текст', fontsize=14,
fontweight='bold', transform=ax.transAxes)
ax.text(0.1, 0.6, 'Курсивный текст', fontsize=12,
style='italic', transform=ax.transAxes)
ax.text(0.1, 0.5, 'Цветной текст', fontsize=12,
color='red', transform=ax.transAxes)
# Текст с рамкой
ax.text(0.1, 0.4, 'Текст в рамке', fontsize=12,
bbox=dict(boxstyle="round,pad=0.3",
facecolor="yellow", alpha=0.7),
transform=ax.transAxes)
# Математические формулы
ax.text(0.1, 0.3, r'$\alpha + \beta = \gamma$',
fontsize=16, transform=ax.transAxes)
ax.text(0.1, 0.2, r'$y = \sin(x)$', fontsize=16,
transform=ax.transAxes)
```

```
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_title('Работа с текстом и шрифтами')
ax.axis('off')
plt.show()
```

12.6 Создание диаграмм и графиков функций

```
import matplotlib.pyplot as plt
import numpy as np
def create_function_plot():
# Создание данных
x = np.linspace(0, 4*np.pi, 1000)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.sin(x) * np.cos(x)
# Создание фигуры с подграфиками
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(10,
12))
# График sin(x)
ax1.plot(x, y1, 'b-', linewidth=2, label='sin(x)')
ax1.set_title('График функции sin(x)')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.grid(True, alpha=0.3)
ax1.legend()
# График cos(x)
ax2.plot(x, y2, 'r-', linewidth=2, label='cos(x)')
ax2.set_title('График функции cos(x)')
ax2.set_xlabel('x')
ax2.set_ylabel('y')
ax2.grid(True, alpha=0.3)
ax2.legend()
# График sin(x) * cos(x)
ax3.plot(x, y3, 'g-', linewidth=2, label='sin(x) *
cos(x)')
ax3.set_title('График функции sin(x) * cos(x)')
ax3.set_xlabel('x')
ax3.set_ylabel('y')
ax3.grid(True, alpha=0.3)
ax3.legend()
plt.tight_layout()
plt.show()
# Запуск функции
create_function_plot()
```

12.7 Работа с изображениями (PIL/Pillow)

```
from PIL import Image, ImageDraw, ImageFont
import numpy as np
# Создание нового изображения
width, height = 800, 600
image = Image.new('RGB', (width, height), 'white')
draw = ImageDraw.Draw(image)
# Рисование примитивов
# Линия
draw.line([(100, 100), (300, 200)], fill='red',
width=3)
# Прямоугольник
draw.rectangle([(100, 250), (300, 350)],
outline='blue', width=2)
# Залитый прямоугольник
draw.rectangle([(350, 250), (550, 350)],
fill='lightblue', outline='blue', width=2)
# Эллипс
draw.ellipse([(100, 400), (300, 500)],
outline='green', width=2)
# Залитый эллипс
draw.ellipse([(350, 400), (550, 500)],
fill='lightgreen', outline='green', width=2)
# Многоугольник
points = [(600, 100), (700, 150), (650, 200), (550,
200), (500, 150)]
draw.polygon(points, outline='purple',
fill='lightpink', width=2)
# Текст
try:
# Попытка использовать системный шрифт
font = ImageFont.truetype("arial.ttf", 24)
except:
# Использование шрифта по умолчанию
font = ImageFont.load_default()
draw.text((100, 50), "Графические примитивы в Python",
fill='black', font=font)
draw.text((100, 300), "Прямоугольник", fill='black',
font=font)
draw.text((350, 300), "Залитый прямоугольник",
fill='black', font=font)
draw.text((100, 450), "Эллипс", fill='black',
font=font)
draw.text((350, 450), "Залитый эллипс", fill='black',
font=font)
```

```
draw.text((500, 250), "Многоугольник", fill='black',
font=font)
# Сохранение изображения
image.save('graphics_primitives.png')
print("Изображение сохранено как
'graphics_primitives.png'")
# Отображение изображения
image.show()
```

12.8 Работа с растровой графикой

```
from PIL import Image, ImageFilter, ImageEnhance
import os
def process_image(input_path, output_path,
format='JPEG', quality=85):
"""Обработка изображения с изменением формата и
качества"""
try:
# Открытие изображения
with Image.open(input_path) as img:
print(f"Исходное изображение: {img.size}, {img.mode},
{img.format}")
# Применение фильтров
# Размытие
blurred = img.filter(ImageFilter.BLUR)
# Увеличение контраста
enhancer = ImageEnhance.Contrast(img)
contrasted = enhancer.enhance(1.5)
# Увеличение яркости
brightness_enhancer = ImageEnhance.Brightness(img)
brightened = brightness_enhancer.enhance(1.2)
# Сохранение в разных форматах
if format.upper() == 'JPEG':
img.save(output_path, 'JPEG', quality=quality)
elif format.upper() == 'PNG':
img.save(output_path, 'PNG')
elif format.upper() == 'BMP':
img.save(output_path, 'BMP')
else:
img.save(output_path)
# Получение информации о файле
file_size = os.path.getsize(output_path)
print(f"Обработанное изображение сохранено:
{output_path}")
print(f"Размер файла: {file_size} байт")
print(f"Формат: {format}, Качество: {quality}")
```

```
return True
except Exception as e:
print(f"Ошибка при обработке изображения: {e}")
return False
# Пример использования
# process_image('input.jpg', 'output.jpg', 'JPEG', 85)
```

12.9 Создание интерактивных графиков

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.widgets import Button, Slider
def create_interactive_plot():
# Создание данных
x = np.linspace(0, 2*np.pi, 1000)
# Создание фигуры и осей
fig, ax = plt.subplots(figsize=(10, 8))
plt.subplots_adjust(bottom=0.25)
# Начальные параметры
initial_freq = 1.0
initial_amp = 1.0
# Создание графика
line, = ax.plot(x, initial_amp * np.sin(initial_freq *
x), 'b-', linewidth=2)
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-2, 2)
ax.grid(True, alpha=0.3)
ax.set_title('Интерактивный график sin(x)')
# Создание слайдеров
ax_freq = plt.axes([0.1, 0.1, 0.3, 0.03])
ax_amp = plt.axes([0.6, 0.1, 0.3, 0.03])
slider_freq = Slider(ax_freq, 'Частота', 0.1, 5.0,
valinit=initial_freq)
slider_amp = Slider(ax_amp, 'Амплитуда', 0.1, 2.0,
valinit=initial_amp)
def update(val):
freq = slider_freq.val
amp = slider_amp.val
line.set_ydata(amp * np.sin(freq * x))
fig.canvas.draw_idle()
slider_freq.on_changed(update)
slider_amp.on_changed(update)
plt.show()
# Запуск интерактивного графика
# create_interactive_plot()
```

12.10 Создание 3D графиков

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
def create_3d_plot():
# Создание данных для 3D поверхности
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = np.sin(np.sqrt(X**2 + Y**2))
# Создание 3D графика
fig = plt.figure(figsize=(12, 8))
# 3D поверхность
ax1 = fig.add_subplot(221, projection='3d')
surf = ax1.plot_surface(X, Y, Z, cmap='viridis',
alpha=0.8)
ax1.set_title('3D Поверхность')
ax1.set_xlabel('X')
ax1.set_ylabel('Y')
ax1.set_zlabel('Z')
# 3D проволочная сетка
ax2 = fig.add_subplot(222, projection='3d')
ax2.plot_wireframe(X, Y, Z, alpha=0.6)
ax2.set_title('3D Проволочная сетка')
ax2.set_xlabel('X')
ax2.set_ylabel('Y')
ax2.set_zlabel('Z')
# 3D контур
ax3 = fig.add_subplot(223, projection='3d')
ax3.contour(X, Y, Z, levels=20)
ax3.set_title('3D Контур')
ax3.set_xlabel('X')
ax3.set_ylabel('Y')
ax3.set_zlabel('Z')
# 2D контур
ax4 = fig.add_subplot(224)
contour = ax4.contour(X, Y, Z, levels=20)
ax4.clabel(contour, inline=True, fontsize=8)
ax4.set_title('2D Контур')
ax4.set_xlabel('X')
ax4.set_ylabel('Y')
plt.tight_layout()
plt.show()
# Запуск 3D графика
# create_3d_plot()
```

12.11 Работа с анимацией

```
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
def create_animation():
# Создание фигуры
fig, ax = plt.subplots(figsize=(10, 6))
# Настройка осей
ax.set_xlim(0, 2*np.pi)
ax.set_ylim(-1.5, 1.5)
ax.grid(True, alpha=0.3)
ax.set_title('Анимированный график sin(x)')
# Создание линии
line, = ax.plot([], [], 'b-', linewidth=2)
def animate(frame):
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x + frame * 0.1)
line.set_data(x, y)
return line,
# Создание анимации
anim = animation.FuncAnimation(fig, animate,
frames=200, interval=50, blit=True)
plt.show()
return anim
# Запуск анимации
# anim = create_animation()
```

12.12 Создание GUI приложения с графикой

```
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg
import numpy as np
class GraphicsApp:
def __init__(self, root):
self.root = root
self.root.title("Приложение для работы с графикой")
self.root.geometry("800x600")
self.create_widgets()
self.create_plot()
def create_widgets(self):
# Панель управления
control_frame = ttk.Frame(self.root)
```

```
control_frame.pack(side=tk.TOP, fill=tk.X, padx=5,
pady=5)
ttk.Button(control_frame, text="sin(x)",
command=self.plot_sin).pack(side=tk.LEFT, padx=5)
ttk.Button(control_frame, text="cos(x)",
command=self.plot_cos).pack(side=tk.LEFT, padx=5)
ttk.Button(control_frame, text="sin(x)*cos(x)",
command=self.plot_sin_cos).pack(side=tk.LEFT, padx=5)
ttk.Button(control_frame, text="Очистить",
command=self.clear_plot).pack(side=tk.LEFT, padx=5)
# Слайдер для частоты
ttk.Label(control_frame,
text="Частота:").pack(side=tk.LEFT, padx=(20, 5))
self.freq_var = tk.DoubleVar(value=1.0)
self.freq_scale = ttk.Scale(control_frame, from_=0.1,
to=5.0, variable=self.freq_var, orient=tk.HORIZONTAL)
self.freq_scale.pack(side=tk.LEFT, padx=5)
# Слайдер для амплитуды
ttk.Label(control_frame,
text="Амплитуда:").pack(side=tk.LEFT, padx=(20, 5))
self.amp_var = tk.DoubleVar(value=1.0)
self.amp_scale = ttk.Scale(control_frame, from_=0.1,
to=3.0, variable=self.amp_var, orient=tk.HORIZONTAL)
self.amp_scale.pack(side=tk.LEFT, padx=5)
def create_plot(self):
# Создание matplotlib фигуры
self.fig, self.ax = plt.subplots(figsize=(8, 4))
self.ax.set_xlim(0, 2*np.pi)
self.ax.set_ylim(-3, 3)
self.ax.grid(True, alpha=0.3)
self.ax.set_title('График функции')
self.ax.set_xlabel('x')
self.ax.set_ylabel('y')
# Встраивание в tkinter
self.canvas = FigureCanvasTkAgg(self.fig, self.root)
self.canvas.get_tk_widget().pack(side=tk.TOP,
fill=tk.BOTH, expand=True)
def plot_function(self, func, label, color):
x = np.linspace(0, 2*np.pi, 1000)
freq = self.freq_var.get()
amp = self.amp_var.get()
y = amp * func(freq * x)
self.ax.plot(x, y, color=color, linewidth=2,
label=f"{label} (freq={freq:.1f}, amp={amp:.1f})")
self.ax.legend()
self.canvas.draw()
def plot_sin(self):
```

```
self.plot_function(np.sin, "sin(x)", "blue")
def plot_cos(self):
self.plot_function(np.cos, "cos(x)", "red")
def plot_sin_cos(self):
self.plot_function(lambda x: np.sin(x) * np.cos(x),
"sin(x)*cos(x)", "green")
def clear_plot(self):
self.ax.clear()
self.ax.set_xlim(0, 2*np.pi)
self.ax.set_ylim(-3, 3)
self.ax.grid(True, alpha=0.3)
self.ax.set_title('График функции')
self.ax.set_xlabel('x')
self.ax.set_ylabel('y')
self.canvas.draw()
def main():
root = tk.Tk()
app = GraphicsApp(root)
root.mainloop()
if __name__ == "__main__":
main()
```

12.13 Форматы растровой графики

Python поддерживает множество форматов изображений:

- **BMP** - без сжатия, высокое качество, большой размер
- **JPEG** - сжатие с потерями, хорош для фотографий
- **PNG** - сжатие без потерь, поддерживает прозрачность
- **GIF** - анимация, ограниченная палитра цветов
- **TIFF** - профессиональный формат, поддерживает слои
- **WebP** - современный формат с хорошим сжатием

Пример работы с разными форматами:

```
from PIL import Image
import os
def convert_image_format(input_path, output_path,
format='JPEG', quality=85):
"""Конвертация изображения в другой формат"""
try:
with Image.open(input_path) as img:
# Конвертация в RGB если необходимо
```

```
if format.upper() in ['JPEG', 'JPG'] and img.mode in
['RGBA', 'LA']:
background = Image.new('RGB', img.size, (255, 255,
255))
background.paste(img, mask=img.split()[-1] if img.mode
== 'RGBA' else None)
img = background
# Сохранение в новом формате
if format.upper() in ['JPEG', 'JPG']:
img.save(output_path, format.upper(), quality=quality,
optimize=True)
else:
img.save(output_path, format.upper())
# Информация о файлах
input_size = os.path.getsize(input_path)
output_size = os.path.getsize(output_path)
compression_ratio = input_size / output_size if
output_size > 0 else 0
print(f"Конвертация завершена:")
print(f"Исходный файл: {input_size} байт")
print(f"Результирующий файл: {output_size} байт")
print(f"Коэффициент сжатия: {compression_ratio:.2f}")
return True
except Exception as e:
print(f"Ошибка конвертации: {e}")
return False
```

Краткие итоги. В лекции были изучены основы работы с графикой в Python. Ознакомились с библиотеками matplotlib, PIL/Pillow для создания графиков и обработки изображений. Рассмотрены способы создания графических примитивов, работы с текстом, создания анимаций и интерактивных графиков.

Контрольные вопросы

1. Какие библиотеки используются для работы с графикой в Python?
2. Как создать простой график функции с помощью matplotlib?
3. Для чего используется библиотека PIL/Pillow?
4. Как нарисовать графические примитивы (линии, прямоугольники, круги)?

5. Как работать с текстом и шрифтами в графике?
6. Расскажите о создании 3D графиков.
7. Как создать анимированный график?
8. Какие форматы изображений поддерживает Python?
9. Как создать GUI приложение с графикой?
10. Приведите пример работы с растровой графикой.
11. Как конвертировать изображения между разными форматами?
12. Что такое интерактивные графики и как их создать?