Муниципальное бюджетное образовательное учреждение

“Лицей Физико-Техническая школа”

Работу выполнил учащийся 10 класса:  
Дмитриев Семён Александрович

Руководитель работы:

Гончаров Никита Николаевич

Создание курса по основам программирования на Python

**Содержание**

[**Введение** 3](#_Toc191658450)

[**Глава 1. Анализ теоретических и практических основ при создании курса по Python** 5](#_Toc191658451)

[1.1. Теоретические предпосылки разработки курса 5](#_Toc191658452)

[1.2. Анализ опыта существующих курсов 7](#_Toc191658453)

[1.3. Выводы по анализу и ориентиры для разработки нового курса 9](#_Toc191658454)

[1.4. Заключение 10](#_Toc191658455)

[**Глава 2. Практические аспекты разработки курса и результаты собственной работы** 11](#_Toc191658456)

[2.1. Цели и задачи практического исследования 11](#_Toc191658457)

[2.2. Процесс и логика разработки учебных модулей 11](#_Toc191658458)

[2.3. Пилотная реализация и апробация курса 13](#_Toc191658459)

[2.4. Результаты собственной работы и ключевые достижения 14](#_Toc191658460)

[2.5. Рекомендации по дальнейшему совершенствованию курса 15](#_Toc191658461)

[2.6. Заключение 16](#_Toc191658462)

# **Введение**

**Проблема**

Курс по основам программирования на Python решает несколько взаимосвязанных проблем:

В настоящее время нет поэтапного введения в ключевые концепции программирования, такие как переменные, типы данных, операторы, условные операторы, циклы, функции, массивы и т.д. Многие люди, желающие освоить программирование, сталкиваются с трудностями в самостоятельном изучении, отсутствием систематизированного подхода и нехваткой практических навыков. Существующие ресурсы часто оказываются слишком сложными для новичков или недостаточно структурированными, что приводит к потере мотивации. Нет доступных, понятных и практико-ориентированных учебных материалов по основам программирования на Python для школьников, студентов и начинающих разработчиков, что приводит к неэффективному освоению языка.

**Актуальность**

Это актуально для людей без опыта программирования, желающих освоить эту востребованную профессию или навык:  
  
• Курс учит разбивать сложные задачи на более мелкие, поддающиеся программированию, и решать их с помощью кода. Это развивает логическое мышление и алгоритмические навыки.  
  
• Владение основами программирования и знание Python значительно увеличивают шансы на успех в поиске работы в IT-сфере или других областях, где востребованы навыки программирования.  
  
• Изучение программирования помогает лучше понять, как работают технологии и программное обеспечение, что полезно в современном цифровом мире.

**Цель**

Цель курса по основам программирования на Python — дать ученикам достаточные знания и навыки для начала работы с Python и дальнейшего развития в области программирования, независимо от их дальнейших профессиональных планов, используя Python как язык обучения.

**Задачи**

Исследовать существующие курсы и выявить недостатки. Систематизировать теорию по основам программирования и поэтапно предоставлять ее ученику. Создание задач, позволяющих закрепить теоретические знания.

**Объект исследования**

Объектом исследования в создании курса является процесс обучения программированию на языке Python.

**Предмет исследования**

Предметом исследования в создании курса по программированию являются факторы, влияющие на эффективность обучения программированию, и разработка оптимальной методики обучения, которая обеспечит достижение поставленных целей курса. Он фокусируется на методике обучения, а не на конкретных языковых конструкциях или библиотеках.

**Методы и приёмы**

На первом этапе следует провести анализ целевой аудитории - это влияет на выбор языка, методов преподавания и сложности материала. Дальше нужно сделать моделирование(конструирование) курса. И только потом предоставить его для обучения.

**Практическая значимость**

Практическая значимость курса по программированию на Python огромна, так как он предоставляет ученикам востребованные навыки, открывает широкие профессиональные перспективы и развивает ценные навыки решения проблем.

**Целевая аудитория**

Этот проект наиболее подходит для учеников старших классов, имеющих базовые навыки работы с компьютером и желающих получить введение в программирование.

# **Глава 1. Анализ теоретических и практических основ при создании курса по Python**

## **1.1. Теоретические предпосылки разработки курса**

**1.1.1. Роль языка Python в современном программировании**

Python был создан Гвидо ван Россумом в конце 80-х — начале 90-х годов и изначально развивался как язык, ориентированный на «читаемость» и простоту написания кода. Со временем эта особенность превратила Python в один из ключевых инструментов для быстрого прототипирования и решения широкого круга задач. Его преимущества:

1. **Лаконичный и понятный синтаксис**  
   Благодаря строгим правилам отступов и отсутствию избыточных конструкций (например, фигурных скобок), читабельность программ на Python чрезвычайно высока. Это способствует более быстрому пониманию кода как новичками, так и опытными разработчиками.
2. **Большое сообщество**  
   Активная поддержка сообщества делает Python удобным для учебных целей: существует множество онлайн-форумов, библиотек, учебников, конференций и курсов. Соответственно, при создании собственного курса можно ориентироваться на проверенные практики, заимствовать примеры и методики.
3. **Обширная стандартная библиотека**  
   Python располагает широким набором встроенных модулей, что позволяет начинающим программистам быстро осваивать практические аспекты программирования (работа с файлами, сетью, данными), не погружаясь в детали низкоуровневых реализаций.
4. **Мультипарадигменность**  
   Язык поддерживает процедурный, объектно-ориентированный и функциональный стили программирования. Для образовательного процесса это означает возможность последовательно знакомить студентов с разными методами организации кода, постепенно усложняя задачи.
5. **Популярность в научной среде**  
   Python востребован в областях машинного обучения, анализа данных и научных вычислений (библиотеки NumPy, SciPy, pandas, TensorFlow и другие). Включение в курс примеров из аналитики или научной сферы может делать обучение более интересным и ориентированным на практику.

**1.1.2. Психолого-педагогические особенности обучения программированию**

Чтобы методически грамотно выстроить курс, необходимо учитывать особенности восприятия новичков:

1. **Наглядность материала**  
   Для большинства начинающих крайне важны примеры реального кода, иллюстрирующие конкретные концепции. Использование интерактивных инструментов (онлайн-редакторов, ноутбуков Jupyter) и визуальных средств (диаграммы, пошаговая анимация выполнения программ) существенно облегчает понимание.
2. **Принцип «от простого к сложному»**  
   Важно постепенно усложнять задачи, закрепляя уже пройденные темы практикой. Слишком резкий переход к абстрактным понятиям или сложным проектам может демотивировать и вызывать непонимание у учащихся.
3. **Модульность**  
   Обучающий материал желательно делить на небольшие понятные блоки (модули или «шаги»), каждый из которых завершается тестом или мини-проектом. Такой формат помогает контролировать прогресс студентов и своевременно корректировать процесс обучения.
4. **Обратная связь**  
   Постоянная интеракция между преподавателем и студентами (через форумы, чаты, систему тестирования) помогает быстро устранять пробелы, а также адаптировать программу к потребностям и уровню группы.
5. **Мотивация через практику**  
   В ходе курса рекомендуется проводить мини-хакатоны, воркшопы, соревнования по программированию. Студенты, которым демонстрируют реальное применение полученных знаний, учатся с большим энтузиазмом.

**1.1.3. Стандарты и рекомендации в сфере обучения Python**

Существует несколько авторитетных организаций и инициатив, формирующих определённые стандарты в преподавании программирования:

* **Python Software Foundation (PSF)** регулярно публикует руководства, статьи и материалы по развитию и применению Python.
* **Всемирные сообщества по Data Science** (например, Kaggle, DataCamp) предлагают разнообразные курсы для разных уровней подготовки, от полного новичка до продвинутых специалистов.
* **Аккредитованные образовательные программы** (в университетах и техникумах) часто включают вводный курс Python в блок дисциплин «Основы программирования». В большинстве программных документов акцент делается на практическую часть (алгоритмы и структуры данных, обработка данных, базовая визуализация и т.д.).

Изучение данных методических рекомендаций и существующих курсов показывает, что следует придерживаться баланса между теорией (базовые конструкции языка, фундаментальные концепции информатики) и практикой (самостоятельное написание и разбор кода).

## **1.2. Анализ опыта существующих курсов**

**1.2.1. Университетские курсы**

В ряде ведущих вузов (например, MIT, Стэнфорд, МФТИ, ИТМО) уже давно применяют Python как язык первого знакомства студентов с программированием. Наиболее характерные черты данных курсов:

1. **Сильная математическая составляющая**  
   Студентам предлагаются задачи на базовые алгоритмы, рекурсию, сортировки, анализ временной и пространственной сложности программ. Акцент делается на том, чтобы учащиеся не только научились писать код, но и понимали внутренние принципы работы алгоритмов.
2. **Гибкая организация занятий**  
   Теоретические лекции комбинируются с лабораторными работами, где студенты пишут реальные программы, консультируясь с преподавателем и ассистентами. Такой формат способствует формированию навыков самостоятельной отладки и поиска ошибок.
3. **Разработка мини-проектов**  
   В конце семестра или учебного модуля студенты обычно готовят проект, в котором применяются все основные темы курса: работа с файлами, сетью, простая визуализация, элементарные структуры данных. Эта практика закрепляет и систематизирует полученные знания.

**1.2.2. Онлайн-школы и платформы интерактивного обучения**

Популярные сайты (Coursera, edX, Stepik, Udemy) предоставляют широкий выбор вводных курсов по Python. Можно выделить несколько общих тенденций:

1. **Разбитие программы на короткие видеолекции**  
   Видеоматериалы длиной 5-10 минут позволяют быстро объяснить конкретную тему (например, «условные операторы»), после чего сразу даётся практическое задание или тест.
2. **Автоматизированная система проверки заданий**  
   Студенты загружают решение на сервер, где специальный скрипт сравнивает его с эталоном (или выполняет тестовые прогоны), позволяя быстро получить результат и рекомендации по исправлению.
3. **Проектная активность**  
   Многие курсы вводят «проектные модули», когда слушатели должны применить весь освоенный материал для создания, например, несложного чат-бота, веб-приложения, скрипта для обработки таблиц в формате CSV или программы для парсинга сайтов.
4. **Сообщество и геймификация**  
   Некоторые платформы активно используют элементы геймификации (ачивки, рейтинги, «очереди лидеров»), а также форумы и чаты для студентов, чтобы повысить вовлечённость и помочь в решении возникающих вопросов.

**1.2.3. Корпоративные тренинги**

В рамках компаний, особенно в IT-сфере или организациях, активно работающих с данными (аналитика, биг дата), часто проводятся внутренние курсы Python для сотрудников. Основной акцент здесь — быстрый старт и решение конкретных бизнес-задач, а также:

1. **Упор на практику**  
   Работникам даётся минимум теоретических сведений, но предлагается множество практических кейсов, отражающих специфику компании (работа с базами данных, автоматизация отчётности и пр.).
2. **Короткие интенсивы**  
   Занятия могут быть организованы в формате «погружения» (2-3 дня подряд по 5-6 часов в день) либо коротких регулярных сессий (по 2 часа несколько раз в неделю). В обоих случаях приоритет — максимально возможная эффективность использования времени.
3. **Наличие наставников**  
   В крупных корпорациях или IT-стартапах наставниками часто выступают опытные разработчики, которые могут индивидуально подсказывать сотрудникам, где искать ошибки, какие библиотеки применять.

## **1.3. Выводы по анализу и ориентиры для разработки нового курса**

1. **Формирование прочного фундамента**  
   Независимо от формата (университет, онлайн-платформа или корпоративное обучение), курс по Python должен начинаться с базовых понятий: переменные, типы данных, операции ввода-вывода, условные конструкции, циклы. Здесь важно уделять внимание качеству объяснения и числу практических примеров, так как именно на этапе знакомства с языком закладываются основные навыки.
2. **Постепенное углубление**  
   Хорошим подходом является принцип спирального обучения: после освоения самых простых аспектов языка (арифметических операций, строк, простых функций) постепенно добавляются темы о структурах данных (списки, словари), обработке ошибок, работе с файлами, простых алгоритмах. В конце курса целесообразно дать начальное понимание о модулях и пакетах, чтобы слушатели умели ориентироваться в экосистеме Python.
3. **Практические задания и проекты**  
   Многочисленный опыт подтверждает: без систематической практики закрепление знаний затруднительно. Поэтому новый курс необходимо разбить на короткие теоретические блоки, после каждого из которых следует обязательная практическая часть. Также логично включить один или несколько мини-проектов — такие проекты позволят студентам увидеть пользующийся спросом «продукт» собственных усилий (например, простое веб-приложение или бот).
4. **Гибкий формат тестирования и проверки**  
   Чтобы обеспечить обратную связь, стоит задействовать либо специализированные онлайн-платформы, либо интегрированные системы проверки заданий. Это позволит автоматизировать проверку большинства задач и разгрузить преподавателя, давая ему время на более глубокий анализ проектов или сложных ошибок.
5. **Учёт разнообразия аудитории**  
   В группу студентов могут входить совершенно разные люди: одни имеют базовые навыки программирования, другие пришли «с нуля», кто-то ориентирован на аналитику данных, а кто-то — на разработку веб-приложений. При создании универсального вводного курса на Python желательно предусмотреть модульность: основные темы обязательны для всех, а дополнительные специализации (например, анализ данных, веб-программирование, скрипты администрирования) могут быть предложены как факультативы.
6. **Поддержка и сопровождение**  
   Наличие канала связи (форум, чат, онлайн-консультации) облегчает процесс обучения. Студенты, получающие быструю и детальную обратную связь, менее подвержены «сгоранию» и лучше осваивают материал.

## **1.4. Заключение**

Анализ теоретических аспектов, а также обзор существующих курсов и методических рекомендаций демонстрируют универсальность и актуальность Python как для начинающих программистов, так и для профессионалов, работающих в различных отраслях. В ходе исследования выявлены основные подходы к обучению, общие трудности и способы их преодоления.  
Важное практическое наблюдение — необходимость совмещения понятного, логичного объяснения базовых конструкций языка с регулярно возобновляющейся практикой программирования. Успешная реализация курса требует тщательного планирования структуры модулей, а также продуманной системы заданий и проектов.

На основании результатов, полученных в данной главе, во второй главе будут сформированы **цели и задачи** курса, определена **целевая аудитория** и представлен **подробный учебный план**, отражающий концепцию, направленную на эффективное усвоение основ Python.

# **Глава 2. Практические аспекты разработки курса и результаты собственной работы**

В первой главе было рассмотрено общее обоснование важности языка Python и методических подходов к его преподаванию. Настоящая глава посвящена **практическим вопросам создания курса**: формированию структуры учебных модулей, отработке методик обучения, выбору заданий и тестов. Здесь обобщён опыт собственной работы над пилотными материалами и описано, как результаты этого опыта легли в основу полноценного учебного плана.

## **2.1. Цели и задачи практического исследования**

При создании курса возникла необходимость проверить, насколько выбранная структура и методы преподавания действительно помогут осваивать Python «с нуля» в максимально короткие сроки и с хорошим уровнем понимания. Исходя из этого, были сформулированы следующие **цели**:

1. **Определить оптимальный набор тем** для базового курса, начиная от элементарного синтаксиса до простых структур данных.
2. **Испытать в деле** различные форматы обучающих материалов — от коротких теоретических заметок до практических мини-проектов — и понять, какие из них наиболее эффективны для освоения языка.
3. **Разработать систему обратной связи** и автоматизированной проверки, чтобы в дальнейшем масштабировать курс и облегчить проверку заданий.
4. **Выявить типовые ошибки** и трудности, возникающие при изучении Python, чтобы заранее включить разъяснения и соответствующие упражнения в основную программу.

Для решения этих задач проводилась **практическая апробация** отдельных модулей курса (далее — «пилотных»), которая позволила внести корректировки в структуру, содержание и методику обучения прежде, чем курс примет окончательный вид.

## **2.2. Процесс и логика разработки учебных модулей**

**2.2.1. Выделение блоков изучаемых тем**

Первым шагом стало **разделение содержания** на несколько блоков (или модулей), соответствующих ключевым концепциям Python. Мы стремились к тому, чтобы каждый модуль был относительно **самодостаточным** и логично завершался практическим заданием.

1. **Блок «Основы синтаксиса и базовые операции»**
   * Понимание, как запустить код Python и работать с файлами **.py**;
   * Типы данных (целые числа, вещественные, строки), операции над ними, простейший ввод-вывод;
   * Короткие упражнения для закрепления синтаксиса (например, «Напишите программу, складывающую два числа»).
2. **Блок «Условные операторы и циклы»**
   * **if/elif/else**, логические выражения;
   * Циклы **for** и **while**, оператор **break/continue**;
   * Наглядные схемы (блок-схемы) для понимания, как работает цикл, и простые алгоритмы (пробежка по диапазону чисел, подсчёт сумм/произведений).
3. **Блок «Работа со структурами данных»**
   * Списки, кортежи, словари: как создавать, изменять, перебирать;
   * Встроенные методы (**append**, **sort**, **get**, **update** и т. д.);
   * Практические задания на анализ и хранение данных (например, мини-проекты по статистике или парсингу текстов).
4. **Блок «Функции»**
   * Основные знания по функциям (вызов возврат)
   * Рекурсия

Такое разбиение легло в основу практического эксперимента по созданию «мини-курса», на основе которого мы впоследствии сформировали **единый учебный план**.

**2.2.2. Структура каждого модуля**

Каждый модуль в пилотном формате содержал следующие компоненты:

1. **Короткая теория**
   * Небольшой текстовый материал или видео (5–10 минут), где объясняются ключевые идеи темы и синтаксис. Важно было сделать упор на **лаконичность** и **доступность** объяснений.
2. **Примеры кода**
   * Для наглядности приводились фрагменты программ, демонстрирующие работу тех или иных конструкций (например, «условие if» или цикл со счётчиком). В пояснениях — пошаговый разбор, что происходит при выполнении.
3. **Практические упражнения**
   * Набор задач, условно разделённых на три уровня:
     + «Разминка» (самые простые, на базовые операции);
     + «Стандартный уровень» (закрепление ключевых умений модуля);
     + «Сложные» (требуют комбинации нескольких идей, часто предполагают небольшой «мини-проект»).
4. **Тесты или контрольные вопросы**
   * Для быстрой самопроверки теоретических знаний. Большая часть таких тестов — с выбором ответа или мелкими фрагментами кода на определение результата.
5. **Рекомендации по дополнительному материалу**
   * Ссылки на документацию Python, официальные руководства и статьи;
   * Советы по сервисам, где можно тренироваться решать короткие задачи (например, Codewars, LeetCode, Py.CheckiO — в зависимости от уровня).

## **2.3. Пилотная реализация и апробация курса**

**2.3.1. Тестовое внедрение модулей**

Для проверки концепции и контента было решено **тестово внедрить** четыре основных модуля (описанных выше) в небольшой пробной среде. На этапе проектирования мы опирались на следующие инструменты:

* **Платформа для размещения материалов** (Stepik), позволяющая создавать последовательные шаги и выкладывать видеоматериалы;
* **Автоматизированная проверка заданий** (система тестов, при загрузке решения проверяющая корректность вывода программы на различных входных данных).

**2.3.2. Наблюдения и коррективы в процессе разработки**

В ходе тестового внедрения выяснилось:

1. **Минимальный порог вхождения** в Python действительно невысок, но важно дать **пошаговую инструкцию** по настройке среды (или ссылку на онлайн-IDE), иначе новички тратят много времени на решения технических вопросов.
2. **Объём теоретического блока** должен оставаться небольшим и сопровождаться примерами. Если теория «насыщена» лишними деталями, у начинающих быстро возникает перегрузка. При этом более сложные аспекты языка (такие как исключения, анонимные функции) целесообразно выносить за пределы «базового» модуля.
3. **Упражнения** должны быть чётко структурированы: от элементарных заданий к более комплексным. В пилотном курсе особенно хорошо сработало правило: «1–2 примера решения в теории + сразу же 3–4 задачи для самостоятельной практики».
4. **Необходима система постепенной проверки**: сначала пользователь выполняет небольшие тесты внутри модуля, потом сдаёт более «объёмное» практическое задание. Это помогает шаг за шагом закреплять материал.

## **2.4. Результаты собственной работы и ключевые достижения**

На основе внедрения пилотных версий модулей были достигнуты следующие результаты, существенно влияющие на дальнейшую разработку курса:

1. **Сформирована базовая «карта» обучения**  
   Четыре основных модуля — «Основы синтаксиса», «Условия и циклы», «Структуры данных» «Функции» — подтверждают свою логику следования.
2. **Разработаны примеры и шаблоны заданий**
   * Банковский модуль задач: «вывести все числа в диапазоне», «посчитать сумму цифр», «выбрать минимальный элемент списка»;
   * Набор небольших проектов: «небольшой калькулятор», «простейший анализ текстового файла»;
   * Контрольные тесты на 10–15 вопросов для быстрой оценки понимания темы.
3. **Наладилась система автопроверки**  
   Для каждого упражнения подготовлен комплект **тестовых сценариев**. Это позволило оперативно видеть, где наиболее частые сбои, и какие части теории нужно дополнительно разъяснять.
4. **Выработаны методические приёмы**
   * Использование псевдокода (блок-схем, схематических описаний алгоритма) перед тем, как писать реальный код;
   * Чёткая структура урока: «теория → примеры → упражнения → проверка → выводы».

## **2.5. Рекомендации по дальнейшему совершенствованию курса**

Исходя из результатов собственной практической работы и полученных наблюдений, можно сформулировать несколько рекомендаций для **полномасштабного** курса:

1. **Модульность и независимые блоки**  
   Курс следует разбить на отдельные «модули-шаги» с возможностью изучать их в гибком порядке (с учётом обязательных базовых тем, конечно). Это позволит адаптировать программу под разные категории слушателей.
2. **Баланс между теорией и практикой**  
   На один теоретический блок должно приходиться как минимум два-три практических задания разного уровня сложности. При этом необходима быстрая проверка (автотесты + поясняющие комментарии).
3. **Визуальные материалы и анимации**  
   Для начинающих ключевую роль играют наглядные схемы. Видео-демонстрации или GIF-анимации, где наглядно видно, как «шаг за шагом» выполняется код, — один из самых мощных инструментов обучения.
4. **Тематические примеры**  
   Желательно включать в курс несколько «сюжетных» заданий (например, обработка данных о продажах, мини-игра или чат-бот), чтобы демонстрировать реальные сценарии использования Python.
5. **Регулярное обновление»**  
   Язык Python и его экосистема развиваются (например, появляются новые возможности в последних версиях, меняются некоторые методы и подходы). Важно предусмотреть механизмы обновления курса — как теоретической части, так и заданий.

## **2.6. Заключение**

Таким образом, практическая работа над пилотными версиями модулей показала высокую востребованность **последовательного и наглядного** изложения материала по Python. Тщательно подобранные примеры, пошаговая подача знаний и система автоматизированной проверки — ключевые факторы успеха. В результате собственной разработки и апробации удалось:

* Выделить **основные модули** курса, в которых логично накапливается сложность;
* Определить **рекомендации** по дальнейшему структурированию (особенно в части циклов и структур данных);
* Создать пул **заданий** и механизм **автотестирования**, позволяющий слушателям сразу проверять свои решения.

Опыт, накопленный на этапе практического исследования, позволит в **следующих главах** более детально представить полный учебный план, включающий как базовые темы, так и дополнительные (объектно-ориентированное программирование, работа с файлами, модули, простая визуализация), а также выстроить систему итоговой проверки знаний и сертификации слушателей.