Ростелеком

3.2.1 Основы Python

Модуль 3. Интеграция приложений и prompt engineering

Оглавление

1. Введение

Здравствуйте!

Добро пожаловать на лекцию, посвящённую основам Python для начинающих, особенно для тех, кто хочет стать хорошим Prompt-инженером. Сегодня мы начнём с самого главного — с понимания, что такое Python и почему он так важен именно в нашей области.

Python — это язык программирования. Он не требует от вас глубоких знаний в информатике, не требует уметь писать сложный код с первого дня, и именно поэтому он идеально подходит для новичков. В то же время, он настолько мощный и гибкий, что используется самыми крупными компаниями в мире для решения серьёзных задач. С помощью Python пишут программы для автоматизации, анализируют данные, управляют нейросетями, обучают модели искусственного интеллекта и, конечно же, создают интерфейсы взаимодействия с такими системами, как ChatGPT.

История Python началась в конце 1980-х годов. Его создатель, Гвидо ван Россум, хотел создать язык, который был бы простым, понятным, но при этом достаточно мощным для серьёзной работы. Название Python он взял не в честь змеи, а по вдохновению от британского комедийного шоу "Monty Python’s Flying Circus". Это многое говорит о духе языка: он не только функциональный, но и человечный, дружелюбный.

Python — это язык, который позволяет быстро писать код и легко его читать. Он понятен даже тем, кто никогда не программировал. И это делает его незаменимым инструментом для специалистов, которые работают с текстами, запросами, данными — то есть именно для вас.

В контексте prompt engineering Python становится своего рода связующим звеном между вами и языковой моделью. Вы можете использовать его, чтобы автоматизировать генерацию запросов, обрабатывать ответы, анализировать результаты. Мы будем использовать его в качестве бекенда для нашего простого веб приложения. Помните, что даже базовые знания Python открывают перед вами совершенно новые горизонты.

2. Как работает код Python

Когда вы пишете код на Python, вы по сути создаёте набор инструкций для компьютера. Эти инструкции записаны в виде обычного текста — вы можете открыть их в любом текстовом редакторе и прочитать. Но для того чтобы компьютер понял, что именно вы хотите от него, нужен особый помощник — интерпретатор Python.

Интерпретатор — это программа, которая читает ваш код построчно и тут же выполняет его. Он словно переводчик, который стоит между вами и компьютером. Вы пишете команды на понятном вам языке — а интерпретатор переводит их компьютеру в тот момент, когда программа запускается. Поэтому даже если вы сделаете ошибку в середине программы, интерпретатор укажет на неё сразу же, как только дойдёт до этой строки.

Здесь важно понять отличие от так называемых компилируемых языков. В таких языках, как C или Java, сначала весь код переводится специальной программой-компилятором в машинный язык — это такой язык, который полностью понятен компьютеру. Только после этого

программа запускается. Этот процесс называется компиляцией и он занимает какое-то время. Зато после компиляции программа работает очень быстро, потому что компьютер уже получил её в "переведённом" виде.

В Python всё происходит прямо в момент запуска — код читается, интерпретируется и сразу же выполняется. Это делает разработку особенно удобной. Вы можете писать несколько строк, запускать их, видеть результат, и сразу же вносить изменения. Это особенно полезно, когда вы экспериментируете с запросами к языковой модели или обрабатываете ответы от неё. Вы можете на лету менять параметры, пробовать разные формулировки и моментально видеть, что получилось.

Интерпретатор Python уже встроен в большинство сред разработки. Вы можете запускать код в Jupyter Notebook, в обычном терминале или даже в браузере через онлайн-среды. Это делает вход в программирование очень мягким и удобным. Вам не нужно настраивать сложную систему — достаточно просто написать пару строк, нажать "выполнить" — и ваш код работает.

3. Переменные и типы данных

Когда вы работаете с данными, вам нужно где-то временно хранить информацию. Для этого в Python используются переменные. Переменная — это просто имя, за которым прячется какое-то значение. Вы можете представить себе переменную как коробку с подписью. Вы пишете, что это, например, “возраст”, и кладёте туда

число 25. В Python это выглядит так: age = 25. Всё просто — слева имя, справа значение.

Вы можете давать переменным любые понятные имена, главное, чтобы они не начинались с цифры и не содержали пробелов. Через переменные вы можете обращаться к данным, изменять их, передавать в функции и получать результаты. Это особенно важно, когда вы работаете с шаблонами prompt'ов, где что-то зависит от вводимых пользователем параметров.

Теперь давайте разберёмся, какие бывают типы данных. Самый простой тип — это целое число, например, 5, 100 или -3. В Python это называется тип int. Если нужно хранить дробные числа, например, 3.14 или 0.5, используется тип float. Когда вы работаете с текстом, то используете строки — это последовательности символов, заключённые в кавычки. Например, "Привет" или "Введите ваш запрос" — это строки, и в Python они обозначаются типом str. Наконец, есть тип bool, который хранит логические значения — True или False. Он используется, например, когда нужно ответить на вопрос "да" или "нет", например, выполнено ли условие, активна ли настройка, совпадают ли значения.

Чтобы понять, какого типа значение хранится в переменной, вы можете воспользоваться функцией type. Например, если вы напишете type(age), Python скажет вам, что это int. Это особенно полезно, когда вы не уверены, с чем имеете дело, или получаете данные из внешних источников, например, из API модели.

Иногда бывает нужно поменять тип. Например, если у вас есть число 5, и вы хотите превратить его в строку, чтобы вставить в текст — вы

можете использовать функцию str(5), и получите строку "5". Или наоборот, если у вас есть строка "42", и вы хотите работать с ней как с числом, вы пишете int("42"), и получаете число 42. Такие преобразования называются приведением типов, и они позволяют вам гибко управлять данными и адаптировать их под нужную задачу.

Всё это — основа, на которой строится вся логика работы с запросами, параметрами и ответами в prompt engineering. Умение понимать типы и использовать переменные — это первый шаг к тому, чтобы ваш код был живым, гибким и полезным.

4. Операции и вычисления

Когда вы начинаете работать с данными, числами или параметрами в запросах, вам рано или поздно нужно будет что-то посчитать. В Python для этого используются стандартные арифметические операции. Сложение обозначается знаком плюс, вы пишете 5 + 3 и получаете 8. Вычитание — это минус, умножение — звёздочка, деление — обычный наклонный слэш. Если вы напишете 10 / 2, вы получите результат 5.0 — это будет число с плавающей точкой, тип float.

Иногда вам нужно целочисленное деление, то есть результат без остатка, просто сколько раз число помещается в другое. Для этого используется двойной слэш. Например, 10 // 3 даст вам 3. Если вас интересует только остаток от деления, используется оператор процента. 10 % 3 даст 1. Это полезно, когда вы проверяете, делится ли что-то на другое без остатка. Возведение в степень обозначается двумя звёздочками. То есть 2 ** 3 — это два в третьей степени, результат 8.

Операторы сравнения позволяют вам задавать вопросы в коде. Например, вы можете спросить, равно ли одно значение другому. Для этого используется двойное равенство — 5 == 5 вернёт True. Если вы хотите проверить, не равны ли значения, используется !=. Знаки больше и меньше работают как в математике. Вы можете написать 7 > 3 или 2 <= 4. Результатом всегда будет True или False, то есть булево значение.

Чтобы создавать более сложные условия, вы можете использовать логические операторы. Оператор and проверяет, выполняются ли оба условия одновременно. Если хотя бы одно из условий ложно, результат будет False. Оператор or проверяет, выполняется ли хотя бы одно из условий. А оператор not просто меняет значение на противоположное. Если not True, то получится False.

Когда вы пишете выражения с несколькими операциями, Python использует приоритет. Сначала выполняются возведение в степень, потом умножение и деление, потом сложение и вычитание. Скобки позволяют вам явно указать порядок действий. Если вы хотите сначала посчитать сумму, а потом умножить, пишите выражение в скобках — как в обычной математике.

Понимание этих операций важно не только для работы с числами. Очень часто вы будете использовать сравнения и логические выражения при анализе входных данных, фильтрации информации, проверке условий в генерации запросов или при создании шаблонов. Это позволяет вам делать ваш код умным, гибким и адаптивным к разным сценариям.

5. Управление потоком программы: условия (if)

Когда вы хотите, чтобы программа вела себя по-разному в зависимости от каких-то условий, вы используете конструкцию if. Это способ сказать Python: если что-то верно, то сделай одно, а если не верно — сделай что-то другое. Например, если пользователь ввёл слово «Да», вы отправляете один prompt, а если «Нет» — другой. Вы пишете: if ответ == "Да": и дальше на следующей строке с отступом — что делать, если условие выполняется.

Если нужно проверить другое условие, используете elif — это сокращение от «else if». То есть: если первое не подошло, проверим второе. И если ни одно условие не сработало, в конце можно написать else — это означает: выполнить код по умолчанию, когда всё остальное оказалось ложным.

Отступы в Python важны. Всё, что должно выполниться внутри if или else, должно быть написано с одинаковым отступом, обычно четыре пробела. Это визуально показывает, что эти строки принадлежат к одному блоку.

Иногда одно условие зависит от другого. Тогда вы можете вкладывать конструкции друг в друга. Например, если внутри одного if вы пишете ещё один if — это называется вложенность. Она позволяет строить сложную логику. Например: если пользователь авторизован, и если у него есть права доступа, то отправить определённый запрос. Такие вложенные проверки помогают строить гибкую структуру принятия решений.

Условия — это как ветвление вашей программы. Вы даёте ей возможность реагировать на ситуацию, а не просто выполнять всё

подряд. В prompt engineering это особенно полезно, когда вы хотите изменить поведение кода в зависимости от типа запроса, параметров модели или структуры ответа. Вы можете строить систему, которая принимает решение как действовать — и всё это начинается с простого if.

6. Управление потоком программы: циклы

Бывают ситуации, когда нужно повторить одно и то же действие несколько раз. Например, сгенерировать серию запросов, обработать список данных или пройтись по набору параметров. Для этого в Python используются циклы. Они позволяют запускать один и тот же блок кода снова и снова — пока не выполнится определённое условие или пока не закончатся элементы, по которым вы проходите.

Цикл for используется, когда вы заранее знаете, сколько раз нужно повторить действие или у вас есть последовательность — например, список слов или чисел. Вы говорите Python: для каждого элемента в этом наборе — сделай вот это. Например, если вы хотите пять раз распечатать разные шаблоны prompt'ов, вы можете использовать цикл for и функцию range. Range — это способ создать последовательность чисел. Когда вы пишете range(5), вы получаете числа от нуля до четырёх. Цикл for перебирает их по одному и на каждом шаге выполняет нужные вам действия.

Цикл while используется в тех случаях, когда вы не знаете точно, сколько шагов потребуется. Он продолжает работать до тех пор, пока выполняется заданное условие. Вы задаёте условие — например, пока

переменная меньше десяти — и код внутри while будет повторяться до тех пор, пока это условие остаётся истинным. Главное — не забыть изменить переменную внутри цикла, иначе вы получите бесконечный цикл, который не остановится сам по себе.

Циклы помогают автоматизировать повторяющиеся действия. Они особенно полезны в prompt engineering, когда вы работаете с массивами данных, перебираете варианты запросов, обрабатываете ответы или создаёте цепочки логики. Вместо того чтобы копировать и вставлять одинаковый код, вы просто пишете один цикл — и он делает за вас всю рутину.

7. Списки и словари

В Python очень удобно работать с группами данных. Для этого есть специальные структуры — списки и словари. Список похож на массив json — это просто набор элементов. Вы можете представить его как коробку, в которой лежат значения: числа, строки, любые другие данные. Списки создаются с помощью квадратных скобок, например [1, 2, 3] или ["текст", "данные", "запрос"]. Вы можете добавлять новые элементы в список с помощью метода append. Например, если у вас есть список prompt'ов, вы можете добавить новый с помощью prompts.append("новый запрос"). Чтобы убрать последний элемент, используется метод pop. Это удобно, когда вы хотите поочерёдно извлекать элементы.

Доступ к элементу списка осуществляется по индексу — то есть по номеру позиции. Индексация начинается с нуля. Если вы хотите

получить первый элемент списка, пишете название списка и в квадратных скобках 0. Например, prompts[0].

Словарь — это структура, где каждой записи соответствует ключ и значение. Вы можете представить себе это как таблицу или карточки, где у каждой карточки есть имя и информация. Словарь создаётся с помощью фигурных скобок. Например: {"роль": "пользователь", "сообщение": "Привет"}. Доступ к данным осуществляется по ключу — вы пишете dict["роль"], и получаете "пользователь". Это очень удобно, когда вы работаете со структурированными данными, например, с сообщениями в формате chat API, где каждое сообщение состоит из ключей role и content.

В цикле for вы можете перебирать элементы списка по одному. Это позволяет, например, проходить по массиву prompt'ов и выполнять для каждого одно и то же действие. Если вы работаете со словарём, то можно перебирать как ключи, так и пары ключ-значение. Для этого словарь имеет специальные методы, такие как items, которые позволяют удобно получать сразу и ключ, и значение.

Списки и словари — это базовые инструменты для хранения, организации и обработки данных. Они особенно важны для prompt-инженеров, потому что позволяют строить гибкие структуры: создавать шаблоны, хранить примеры, обрабатывать параметры и организовывать работу с запросами к модели. Умение пользоваться этими структурами делает вашу работу эффективнее и даёт больше свободы при создании сложных сценариев взаимодействия с языковыми системами.

8. Функции

Функции нужны тогда, когда вы хотите собрать повторяющийся фрагмент кода в одно место и давать ему имя. Это как если бы вы упаковали блок действий в отдельную коробку с ярлыком и могли запускать её в любом месте программы, просто называя это имя. Функции позволяют сделать код компактным, понятным и удобным для повторного использования.

Чтобы создать функцию в Python, используется ключевое слово def. Вы пишете def, затем имя функции, круглые скобки и двоеточие. Всё, что будет внутри этой функции, пишется с отступом. Когда вы хотите вызвать функцию, вы просто пишете её имя и добавляете скобки. Если вы в скобках передаёте какие-то значения — это аргументы. Они позволяют сделать функцию более гибкой. Например, вы можете передать в неё текст prompt'a или параметры генерации, и функция выполнит действия с ними.

Функция может вернуть результат. Для этого используется ключевое слово return. Как только Python встречает return, он выходит из функции и возвращает указанное значение. Вы можете сохранить это значение в переменную и использовать дальше в коде.

Функции особенно полезны в prompt engineering, потому что позволяют отделить логику генерации, обработки, форматирования запросов и ответов от основной части программы. Вы можете создавать шаблоны, которые принимают переменные, генерировать текст, анализировать отклики и всё это упаковывать в отдельные функции. Это делает ваш код не только аккуратным, но и легко масштабируемым.

9. Классы

Давайте разберем, как работают классы в Python на примере нашего сервиса для работы с языковой моделью. Представьте, что класс – это как чертёж или шаблон для создания объектов. Когда мы создаем класс LLMService, мы определяем, какими свойствами и поведением будут обладать все его экземпляры.

В нашем примере есть специальный метод **init** – это конструктор, который автоматически вызывается при создании нового объекта. Обратите внимание, что конструктор принимает параметр sys_prompt – это значит, что при создании объекта мы должны передать ему системный промпт, который определит поведение модели.

Ключевое слово self – это как “я” для объекта. Когда мы пишем self.client или self.sys_prompt, мы создаем переменные, которые будут принадлежать каждому конкретному объекту нашего класса. Это как личные вещи у человека – у каждого своя одежда, свои документы, так и у каждого объекта класса будут свои значения этих переменных.

В нашем конструкторе мы создаем подключение к API через self.client и сохраняем переданный системный промпт в self.sys_prompt. Это позволяет нам гибко настраивать поведение каждого экземпляра класса – мы можем создать несколько объектов с разными системными промптами.

В методе chat мы видим, как используются эти переменные через self. Когда мы обращаемся к self.client, мы используем именно то подключение к API, которое было создано при создании конкретного объекта. А self.sys_prompt позволяет нам использовать заранее заданный стиль общения, который мы передали при создании объекта.

Давайте представим это на простом примере. Если мы создадим два объекта нашего класса с разными системными промтами:

service1 = LLMService(“Ты оператор техподдержки”)

service2 = LLMService(“Ты креативный писатель”)

У каждого из них будет свой системный промпт, хотя код класса один и тот же. Это как два одинаковых автомобиля с разными номерами и владельцами – модель одна, а машины разные.

Такой подход позволяет нам создавать гибкие инструменты для работы с языковой моделью, где базовые настройки выполняются при создании объекта, а мы можем просто использовать готовый функционал. При этом каждый объект может иметь свои уникальные настройки, сохраняя общую логику работы класса.

На практическом занятии мы еще потренируемся создавать и использовать объекты класса – это поможет вам лучше понять, как работают конструкторы и self в Python.

10. Работа с текстом в Python

Текст в Python представлен строками. Это один из самых важных типов данных для prompt-инженера, потому что большая часть взаимодействия с языковыми моделями — это работа с текстами. Со строками можно делать множество операций. Например, вы можете объединять их. Это называется конкатенация. Если у вас есть две строки, вы просто складываете их с помощью знака плюс. Допустим, "Привет, " + "мир!" даст "Привет, мир!". Так можно собирать сообщения, шаблоны, ответы.

Но ещё удобнее использовать f-строки. Это особый способ подставлять значения прямо внутрь текста. Вы пишете перед строкой букву f, а внутри фигурных скобок вставляете переменные. Например, f"Ваш результат: {score}" создаст строку, в которую подставится значение переменной score. Это очень удобно, когда вы создаёте prompt с параметрами, которые заранее неизвестны.

Для обработки текста строки имеют множество встроенных методов. Метод lower делает все буквы в строке маленькими. Это полезно, когда вы хотите сравнивать текст, не обращая внимания на регистр, например, "Да" и "да" будут одинаковыми после преобразования. Метод split разбивает строку на части по какому-то разделителю. По умолчанию — по пробелам. Если у вас есть строка из нескольких слов, split превратит её в список. Метод replace заменяет одну часть строки на другую. Вы можете, например, заменить слово "ошибка" на "внимание" или просто удалить ненужный символ.

Эти инструменты позволяют гибко управлять текстами. А так как в prompt engineering вы постоянно работаете с текстовыми шаблонами, пользовательским вводом и ответами моделей, умение быстро и удобно обрабатывать строки даёт вам возможность делать это эффективно и без лишней сложности.

11. Работа с файлами

Теперь разберем работу с файлами в Python, что особенно важно для prompt инженеров, так как часто приходится сохранять или загружать данные. Для работы с файлами используется функция open(). Она принимает два основных аргумента: путь к файлу и режим

открытия. Режимы бывают разные: "r" для чтения, "w" для записи (если файла нет, он создается, а если есть — перезаписывается), "a" для добавления в конец файла без перезаписи и "x" для создания нового файла.

После открытия файла его нужно прочитать или записать данные. Для чтения используется метод read(), который возвращает все содержимое файла в виде строки. Если нужно прочитать файл построчно, можно использовать readline() или readlines().

Для записи данных в файл применяется метод write(). Важно помнить, что после работы с файлом его нужно закрыть методом close(), чтобы избежать утечек ресурсов. Однако удобнее использовать конструкцию with, которая автоматически закрывает файл после завершения блока кода.

Формат .txt — это простой текстовый файл, с которым легко работать через стандартные методы read() и write(). JSON — более структурированный формат, удобный для хранения словарей и списков. Для работы с JSON в Python есть модуль json. Чтобы прочитать JSON-файл, используйте json.load(), а для записи — json.dump().

Не забывайте проверять пути к файлам и обрабатывать возможные ошибки, например, FileNotFoundError. Это поможет избежать неожиданных проблем при работе с файловой системой.

12. Использование библиотек

В Python для расширения функциональности используются библиотеки, которые подключаются через ключевое слово import. Это

позволяет использовать готовые решения вместо написания кода с нуля. Например, стандартная библиотека math предоставляет математические функции: синусы, косинусы, логарифмы и другие операции, которые могут пригодиться при обработке данных.

Для более сложных вычислений, особенно в задачах машинного обучения и анализа данных, часто применяют библиотеку numpy. Она позволяет эффективно работать с многомерными массивами и матрицами, выполнять быстрые математические операции и применять статистические методы.

Если вы работаете с облачными сервисами, например, ChatGPT, вам может понадобиться библиотека openai. Эта оригинальная библиотека предоставляет удобные инструменты для интеграции машинного обучения и обработки естественного языка в ваши проекты. Этом могут быть модели от компании OpenAI или Yandex Foundation Models.

Чтобы установить сторонние библиотеки, используется менеджер пакетов pip. Например, команда pip install openai загрузит и установит ее на ваш компьютер. После этого библиотеку можно импортировать в код и использовать её функции.

Важно следить за версиями библиотек и документацией, так как API иногда меняется. Также стоит учитывать зависимости между пакетами, чтобы избежать конфликтов. Если вам нужны специфические функции, всегда проверяйте, есть ли готовая библиотека — это сэкономит время и упростит разработку.

13. Заключение

Сегодня мы прошли ключевые концепции Python, ваш фундамент в работе с кодом.

Мы разобрали, как Python превращает написанный код в выполняемую программу: от текста в редакторе до байт-кода и интерпретации. Вы узнали, как создавать переменные и работать с разными типами данных — это основа для хранения и обработки информации.

Мы упомянули операции и вычисления, без которых невозможна ни одна программа. Освоили управление потоком выполнения через условия if и циклы.

Вы познакомились со списками и словарями — мощными структурами данных для организации информации. Узнали, как создавать функции и классы, что делает код более компактными и понятным.

Увидели код, который позволяет работать с текстом, делать чтение и запись файлов в форматах .txt и .json. Это особенно важно для prompt инженера, где обработка строк встречается постоянно.

Наконец, вы узнали, как подключать и использовать библиотеки, такие как openai, чтобы расширять возможности Python и решать сложные задачи в несколько строк кода.

Теперь у вас есть базовый инструментарий, чтобы понимать, как Python используется в prompt engineering. Вникайте в готовые примеры и не бойтесь их изменять, экспериментируйте. Ведь только практика даст вам самые ценные навыки решения разных задач.

Спасибо за внимание!