# Занятие: Введение в Data Science и Python

**Дата:** 6 Января  
**Тема:** Что такое Data Science и Введение в Python

### 🎯 Цели занятия

1. **Понять, кто такой Дата Сайнтист:**
   * Его роль в команде и отличия от других Дата-специалистов (Data Engineer, Data Analyst).
2. **Изучить жизненный цикл проекта:**
   * Этапы работы над данными.
   * Повседневные задачи специалиста.
3. **Роль Python в экосистеме данных:**
   * Почему Python?
4. **«Язык Слизерина»: Основы синтаксиса:**
   * Как и что писать на Python.
   * **Переменные** и **типы данных** (int, float, str, bool).
   * Работа с выводом данных (print) и форматирование строк.

### 📋 Правила урока

* 🤫 **Не перебивать:** Материал подается блоками.
* ⏸ **Вопросы:** В конце каждого блока будут паузы для обсуждения и ответов на вопросы.
* 💬 **Чат:** Если вопрос срочный или боитесь забыть — пишите в чат, я регулярно его проверяю.
* ⏱ **Длительность:** Занятие рассчитано на **~2 часа** с перерывом в 10 минут в середине.
* 📝 **Обратная связь:** В конце занятия будет **QR-код** для ваших отзывов.

# ЦЕЛЬ 1. Понять, кто такой Дата Сайнтист

## 👥 Роли в современной Data-команде

В современной компании работа с данными разделена между несколькими ключевыми специалистами. Каждый отвечает за свой этап превращения «сырой» информации в ценные бизнес-решения.

### 🔍 Data Analyst (Дата-аналитик)

*«Детектив» и мост между данными и бизнесом.*

* **Задача:** Отвечать на вопросы бизнеса (например, *«Почему упали продажи?»*).
* **Что делает:** Собирает данные, очищает их и визуализирует отчеты.
* **Навыки:** SQL, Excel, PowerBI / Tableau.
* **Фокус:** **Дескриптивная аналитика** — описание того, что происходит сейчас или происходило в прошлом.

### 🏗️ Data Engineer (Дата-инженер)

*Архитектор «водопровода» для данных.*

* **Задача:** Доставка данных из систем-источников в хранилище.
* **Что делает:** Строит автоматизированные пути (**pipelines**) и работает с процессами **ETL** (Extract, Transform, Load).
* **Результат:** Организованное хранилище (например, архитектура *Medallion*: Bronze, Silver, Gold).
* **Инструменты:** Python, SQL, Spark, Airflow.

### 📊 BI-analyst / BI Developer (BI-аналитик)

*Мастер автоматизации и визуализации.*

* **Задача:** Создание «единой точки истины» для всей компании.
* **Что делает:** Превращает разовые отчеты в интерактивные дашборды, доступные 24/7.
* **Зачем нужен:** Чтобы все отделы видели одинаковые, актуальные цифры.

### 🧪 Data Scientist (Дата-сайентист)

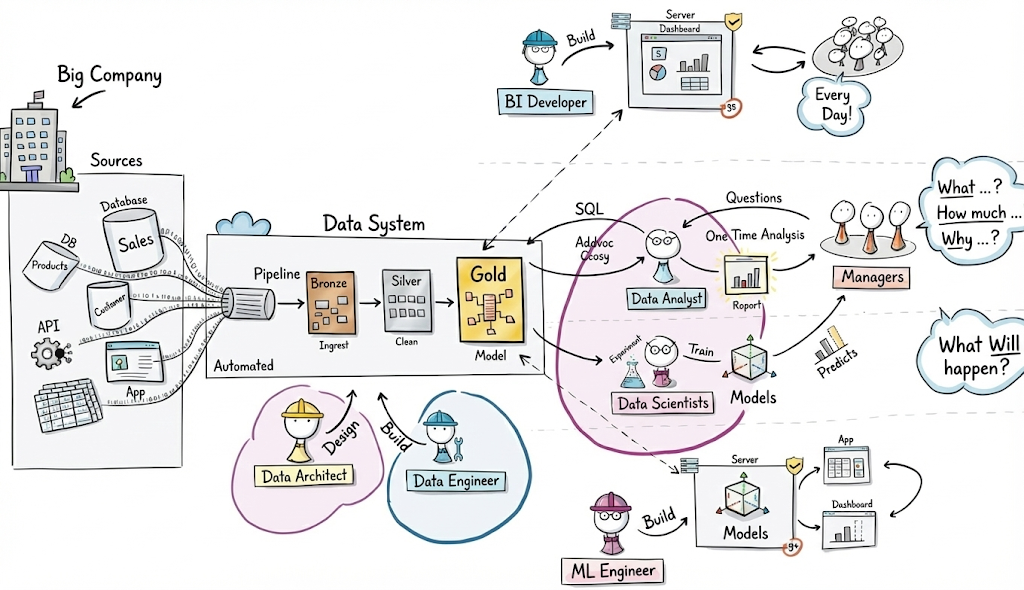
*Исследователь и предсказатель.*

* **Задача:** Ответы на вопросы о будущем (например, *«Кто из клиентов уйдет?»*).
* **Что делает:** Проводит исследовательский анализ (**EDA**), отбирает признаки (**feature engineering**) и обучает модели.
* **Фокус:** **Предиктивная аналитика** — предсказание будущего с помощью математических алгоритмов.

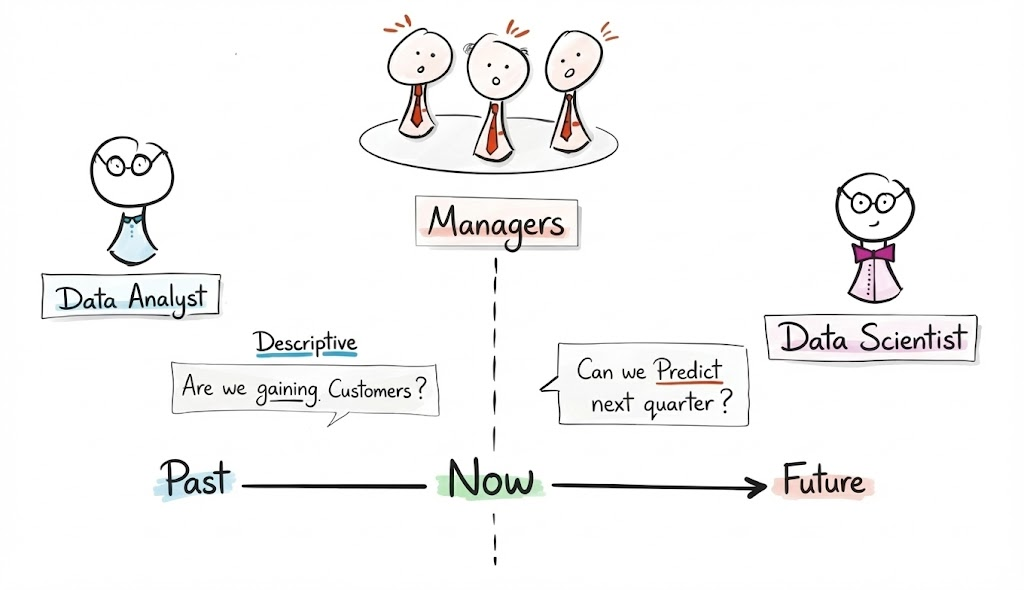
### ⚙️ Machine Learning (ML) Engineer (ML-инженер)

*Связующее звено между кодом и продуктом.*

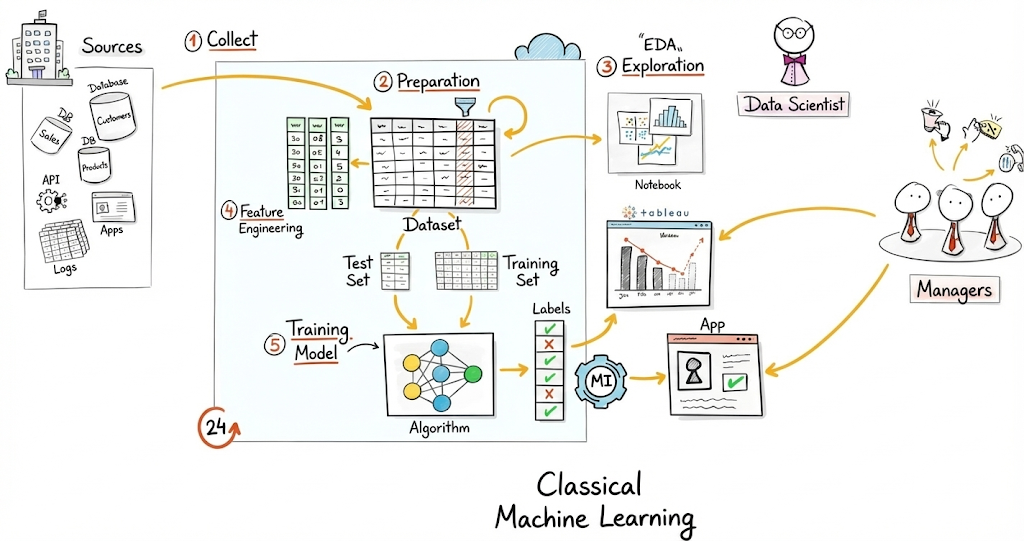
* **Задача:** Перенос «черновика» дата-сайентиста в рабочую среду (Production).
* **Что делает:** Делает модель доступной через **API**, следит за скоростью работы и масштабируемостью.
* **Задачи:** Автоматическое переобучение моделей и интеграция их в реальные приложения.



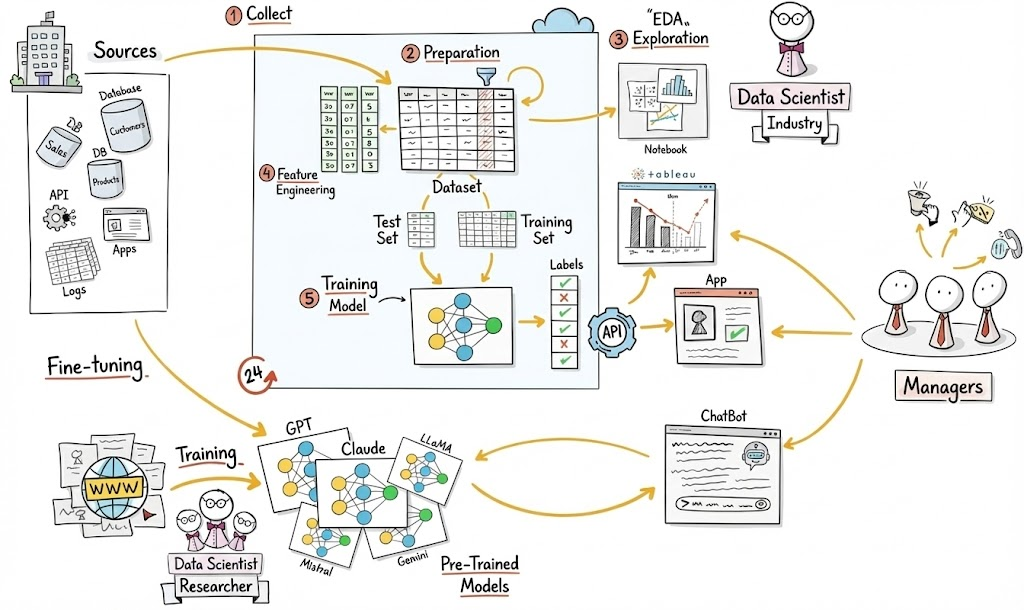
Source: [**The Data Movie | Data Literacy Explained Visually**](https://www.youtube.com/watch?v=J2rQTJby8XM&t=3s) at 13.00



Source: [**The Data Movie | Data Literacy Explained Visually**](https://www.youtube.com/watch?v=J2rQTJby8XM&t=3s) at 15.52



Source: [**The Data Movie | Data Literacy Explained Visually**](https://www.youtube.com/watch?v=J2rQTJby8XM&t=3s) at 24.03



Source: [**The Data Movie | Data Literacy Explained Visually**](https://www.youtube.com/watch?v=J2rQTJby8XM&t=3s) at 26.28

# ЦЕЛЬ 2. Изучить жизненный цикл проекта

## 🧪 Основные обязанности Data Scientist

На основе жизненного цикла проекта можно выделить ключевые области ответственности дата-сайентиста:

### 1. Коммуникация с бизнесом

* **Понимание задачи:** Участие в митингах для выявления болей бизнеса.
* **Постановка гипотез:** Превращение бизнес-запросов в конкретные технические задачи.

### 2. Сбор и обработка данных

* **Ingest & Clean:** Извлечение данных из баз (SQL, API, файлы) и их очистка от «мусора».
* **Pipelines:** Создание автоматизированных путей обработки, чтобы данные были пригодны для анализа.

### 3. Исследование данных (EDA)

* **Exploratory Data Analysis:** Глубокое изучение структуры данных с помощью Python-ноутбуков.
* **Поиск закономерностей:** Визуализация распределений и корреляций для понимания природы данных.

### 4. Разработка и обучение моделей

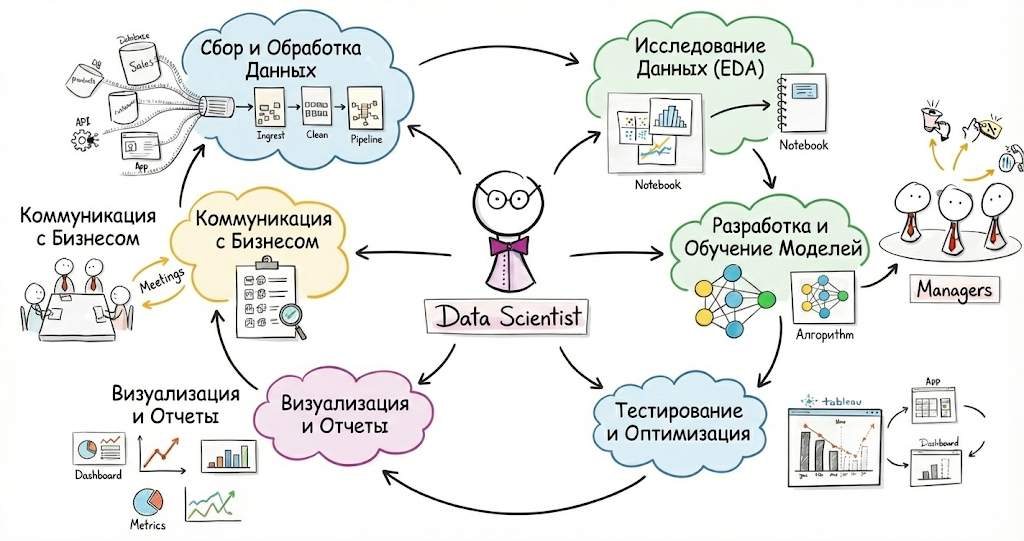
* **Выбор алгоритмов:** Подбор математических моделей (нейронные сети, деревья решений и др.).
* **Training:** Обучение моделей на подготовленных данных для решения задач классификации, регрессии или кластеризации.

### 5. Тестирование и оптимизация

* **Валидация:** Проверка точности модели на новых данных.

### 6. Визуализация и отчеты

* **Презентация результатов:** Создание дашбордов (например, в PowerBI) и визуализация метрик.
* **Интерпретация:** Объяснение менеджерам и стейкхолдерам, как полученные результаты помогут компании.



Made by Nano Banana

# ЦЕЛЬ 3. Роль Python в экосистеме данных

## 🐍 Почему Python — выбор №1 в мире данных?

Python доминирует в сфере Data Science и за её пределами благодаря сочетанию простоты и невероятной мощности.

### 1. Простота и читаемость (High-Level Language)

Python — это язык **высокого уровня**. Это значит, что он создан для людей, а не для машин.

* Его синтаксис логичен и близок к обычному английскому языку.
* Он служит «мостом», который скрывает сложность машинного кода (нулей и единиц), позволяя вам сосредоточиться на решении задачи, а не на архитектуре процессора.

### 2. Эффективность написания кода

Python позволяет делать больше, тратя меньше времени:

* Задачи, которые в Java или C++ занимают десятки строк, в Python часто решаются **одной-двумя строками**.
* Это позволяет аналитикам и разработчикам быстро проверять идеи и создавать сложные приложения с минимальными усилиями.

### 3. Ведущая экосистема для AI и данных

Python — фундамент технологий будущего. Почти все прорывы в области ИИ (например, **ChatGPT** или автопилоты) созданы на Python благодаря его библиотекам:

* **Data Science:** Pandas, NumPy, Plotly (анализ и визуализация).
* **Machine Learning:** PyTorch, TensorFlow, Transformers (создание умных систем).
* **Data Engineering:** Spark и Airflow (автоматизация потоков данных).

### 4. Огромное сообщество

Вы никогда не останетесь один на один с проблемой:

* Тысячи разработчиков делятся знаниями на GitHub и Stack Overflow.
* Если вы столкнулись с ошибкой, скорее всего, кто-то в мире уже её решил и выложил решение в сеть.

### 5. Универсальность и востребованность

Python используется везде: от разработки игр и веб-сайтов до автоматизации роботов.

* Это один из самых востребованных навыков на рынке труда.
* Навыки Python требуются в самых разных отраслях: финансы, медицина, производство и ритейл.

### 6. Автоматизация исполнения

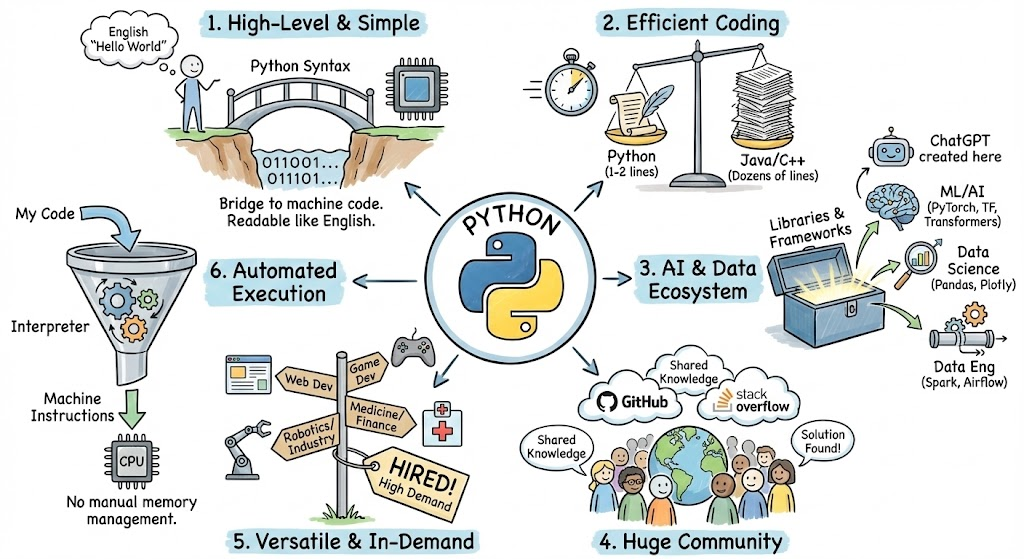
Процесс запуска кода максимально упрощен:

* Специальный инструмент (**интерпретатор**) автоматически превращает ваш код в понятные компьютеру инструкции.
* Вам не нужно заботиться о том, как именно компьютер управляет памятью или процессором «под капотом».

### 🚗 Аналогия для понимания

Представьте, что языки программирования — это средства передвижения:

* **Низкоуровневые языки (как C):** Это как собирать двигатель автомобиля по винтику, прежде чем поехать. Это дает мощь, но требует глубоких знаний механики.
* **Python:** Это как сесть в современный автомобиль и просто повернуть ключ зажигания. Сложные механизмы уже работают за вас, позволяя вам сосредоточиться исключительно на том, **куда** вы хотите доехать.



Made by Nano Banana

# ЦЕЛЬ 4. «Язык Слизерина»: Основы синтаксиса

### - Переменные

### - Базовые типы данных

### - Print statement

## 🐍 Переменные в Python

**Переменная** — это именованная область памяти, в которой хранится определённое значение. Простыми словами, это «коробка» с наклейкой (именем), внутрь которой мы кладём данные.

## 📥 Как создать переменную?

В Python не нужно заранее объявлять тип данных — мы просто используем знак **=** для присваивания значения.

# Пример создания переменных  
name = "Harry" # Строка (str)  
power\_level = 9000 # Целое число (int)  
is\_wizard = True # Булево значение (bool)

## 📜 Правила именования (PEP 8)

Чтобы код был понятным и профессиональным, придерживайтесь следующих правил:

### 🧩 1. Стиль snake\_case

Если имя состоит из нескольких слов, разделяйте их нижним подчёркиванием.

**✅ Правильно:**

user\_age  
total\_score  
is\_student\_active

**❌ Неправильно:**

userAge  
UserAge  
user-age

### 🔤 2. Только латиница

Используйте английские буквы.

**✅ Правильно:**

price = 100

**❌ Неправильно (хотя Python это поймёт):**

цена = 100

В профессиональной среде это считается плохой практикой.

### 🔢 3. Начинайте с буквы

Имя переменной не может начинаться с цифры.

**✅ Правильно:**

player1  
step\_2

**❌ Неправильно:**

1st\_place

### 🔠 4. Регистр имеет значение

Age и age — это **две разные переменные**.

### ⛔ 5. Запрещённые слова

Нельзя использовать зарезервированные слова Python:

print  
if  
while  
True

В Data Science мы часто работаем с огромными таблицами и сложными формулами. Если назвать переменную просто x или a, уже через 10 минут будет непонятно, что в ней хранится.

### 💡 Золотое правило

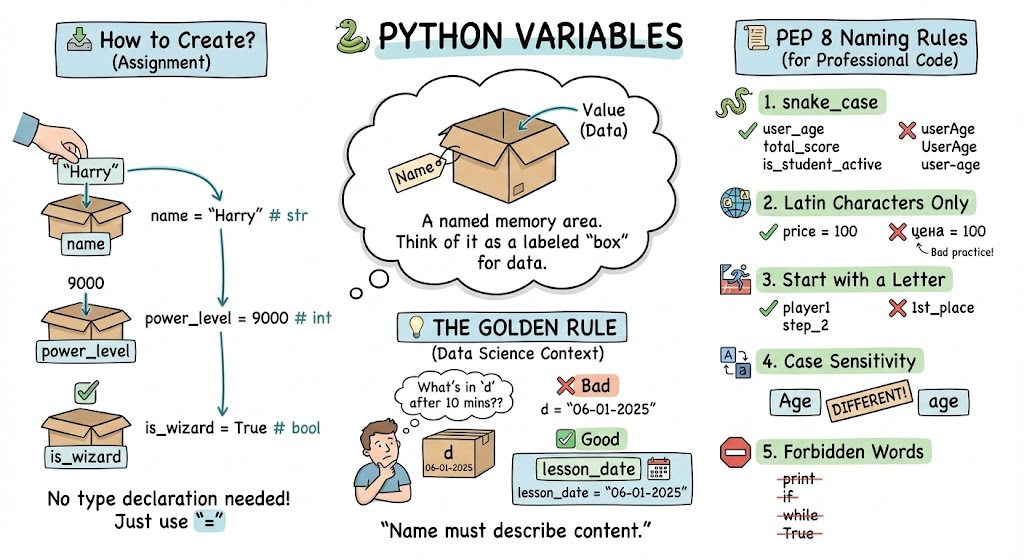
**Имя переменной должно описывать её содержимое.**

**❌ Плохо:**

d = "06-01-2025"

**✅ Хорошо:**

lesson\_date = "06-01-2025"



Made by Nano Banana

## 📊 Основные типы данных в Python

В Python **всё является объектом**, и каждый объект имеет свой тип. Для дата-сайентиста понимание типов данных — это как знание ингредиентов для зельеварения: если перепутать число со строкой, код «взорвётся».

## 1️⃣ Числовые типы (Numbers)

### 🔢 int (integer)

**Целые числа.** Используются для счёта объектов (количество строк в таблице, количество учеников).

students\_count = 30

### 🔢 float (floating point)

**Числа с плавающей точкой** (десятичные дроби). Используются для точных измерений: вероятности, средние значения, цены.

pi\_value = 3.14159

⚠️ **Важно:** в Python в качестве разделителя используется **точка**, а не запятая.

## 2️⃣ Строки (Strings — str)

Это **текстовые данные**. Строки всегда заключаются в кавычки — одинарные ' или двойные ".

academy = "Hogwarts"

### ➕ Сложение строк (конкатенация)

first\_name = "Harry"  
last\_name = "Potter"  
full\_name = first\_name + " " + last\_name  
# Результат: "Harry Potter"

## 3️⃣ Логический тип (Boolean — bool)

Принимает всего **два значения**:

* True — Истина
* False — Ложь

Используется для **проверок и условий**.

is\_python\_easy = True

⚠️ **Внимание:** True и False всегда пишутся **с большой буквы**.

## 🛠️ Как узнать тип данных?

Если вы не уверены, что лежит в переменной, используйте встроенную функцию type() — это ваш **«детектор заклинаний»**.

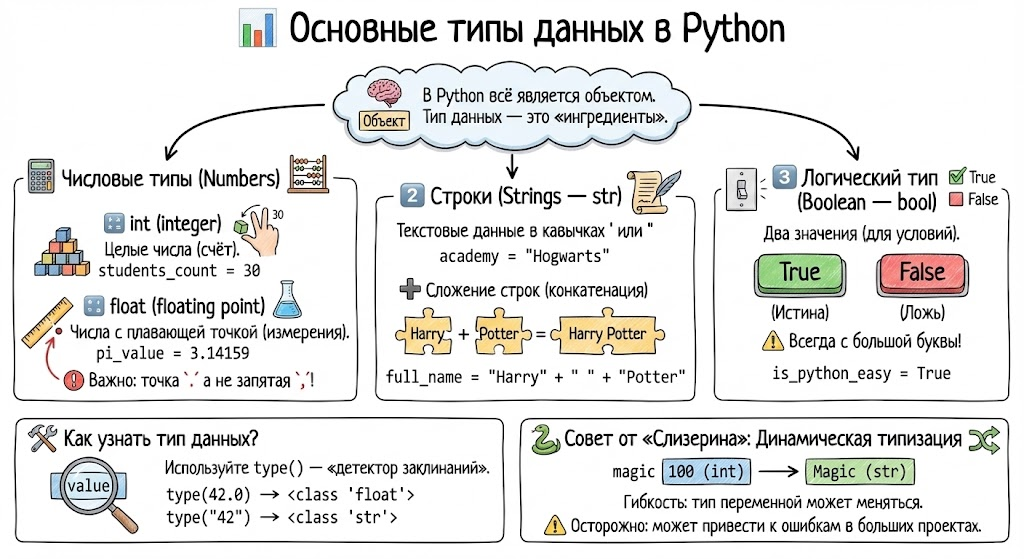
# Попробуйте запустить это в Google Colab  
value = 42.0  
print(type(value)) # <class 'float'>  
  
value = "42"  
print(type(value)) # <class 'str'>

## 🐍 Динамическая типизация

Python очень гибкий язык. В одну и ту же переменную можно сначала положить число, а потом строку.

magic = 100 # Сейчас это int  
magic = "Magic" # А теперь это str!

⚠️ **Будьте осторожны:** в больших проектах это часто приводит к трудноуловимым ошибкам.



Made by Nano Banana

## 🏷️ Аннотации типов (Type Hints): подсказки vs законы

В Python запись вида name: str = "Alice" — это **не жёсткое ограничение**, а **Type Hint** (подсказка типа). Разберёмся, как это работает на практике.

## 1️⃣ Подсказки — это не законы

В таких языках, как **Java** или **C++**, тип — это «жёсткая клетка»: если вы объявили переменную числом, вы **никогда** не сможете положить туда текст.

В **Python** тип — это скорее **этикетка на коробке**:

* вы можете приклеить этикетку «Яблоки» (: str),
* но положить внутрь кирпич (= 5).

Python — язык «для взрослых»: он **доверяет вам**, но предупреждает, что путаница с этикетками может привести к проблемам.

### ❗ Пример «обмана» аннотации

# Мы обещаем, что тут будет строка  
age\_info: str = "Двадцать"  
  
# Но Python спокойно позволяет положить число  
age\_info = 20  
  
print(age\_info) # Код сработает, ошибки не будет

## 2️⃣ Зачем мы их используем? (три кита)

Несмотря на то, что Python позволяет нарушать аннотации, **профессионалы всегда их пишут** по трём причинам.

### 🚀 A) Автодополнение (IntelliSense)

Редактор кода (VS Code, PyCharm, Colab) видит аннотацию и **понимает, что внутри переменной**.

Если указано : str, редактор сразу предложит методы строк:

.upper()  
.lower()  
.split()

➡️ Это ускоряет написание кода и защищает от опечаток.

### 📖 B) Читаемость (код как документация)

Код читают **в 10 раз чаще**, чем пишут. Через месяц вы можете забыть:

* принимает ли функция одно имя,
* или список ID,
* или вообще словарь.

Аннотация — это **документация, которая всегда перед глазами**.

### 🛡️ C) Внешний контроль (mypy)

В больших компаниях (Google, Netflix, Яндекс) код проверяют **статические анализаторы** — например, mypy.

Если вы пообещали str, а положили int, mypy:

* подсветит это как ошибку
* и не даст отправить код в продакшен

## 🧪 Сравнение типов (для закрепления)

* **Целое число** → count: int → 10
* **Дробное число** → price: float → 99.99
* **Текст** → user: str → "Harry"
* **Логика** → is\_valid: bool → True

## 💡 Итог

**Аннотация типов — это договор с самим собой и командой.**

Python не накажет вас за его нарушение мгновенно, но его соблюдение делает код **профессиональным, понятным и надёжным**.

## ОБРАТНАЯ СВЯЗЬ

<https://tally.so/r/xXrM1J>



## 🧩 Дополнительные необязательные: 3 задачи

Темы **строго ограничены** тем, что мы уже прошли:

* переменные
* типы данных (int, float, str, bool)
* type()
* динамическая типизация
* аннотации типов (Type Hints)

❗ **Запрещено:** if, for, функции, списковые включения, любые конструкции управления.

## 🧪 Задача 1. Иллюзия чисел

Дан код:

value1: str = "100"  
value2: int = 100  
value3: float = 100.0  
value4: bool = True

**Задание:**

1. Выведите значение и тип **каждой** переменной с помощью print() и type().
2. Объясните (словами):
   * какие значения выглядят одинаково,
   * но на самом деле имеют **разные типы**.
3. Ответьте: почему True может быть неожиданным в числовых вычислениях?

## 🧪 Задача 2. Переменная-хамелеон

Дана одна переменная с аннотацией:

magic: int = 42

**Задание:**

1. Последовательно присвойте в magic:
   * строку с тем же значением
   * дробное число
   * логическое значение
2. После **каждого присваивания** выведите:

Значение: <value> | Тип: <type>

1. Ответьте письменно:
   * нарушена ли аннотация?
   * почему Python не считает это ошибкой?

## 🧪 Задача 3. Обманчивая документация

Посмотрите на код:

user\_age: int = "18"  
average\_score: float = "89.5"  
is\_active: bool = "True"

**Задание:**

1. Для каждой переменной выведите её **реальный тип**.
2. Перепишите код так, чтобы:
   * аннотации **соответствовали реальным типам** значений,
   * **или** значения соответствовали аннотациям.
3. Ответьте:
   * кто отвечает за соблюдение аннотаций: Python или программист?

## ✅ Ответы к задачам

Темы:

* переменные
* типы данных (int, float, str, bool)
* type()
* динамическая типизация
* аннотации типов (Type Hints)

## 🧪 Ответы к задаче 1. Иллюзия чисел

Исходный код:

value1: str = "100"  
value2: int = 100  
value3: float = 100.0  
value4: bool = True

### Реальные типы значений:

* type(value1) → <class 'str'>
* type(value2) → <class 'int'>
* type(value3) → <class 'float'>
* type(value4) → <class 'bool'>

### Пояснение:

* Значения "100", 100 и 100.0 выглядят похоже, но имеют **разные типы**:
  + строка,
  + целое число,
  + дробное число.
* True — это **логический тип**, а не строка и не обычное число.

### Почему True опасен в числах:

В Python True ведёт себя как 1, а False как 0. Поэтому случайное участие bool в вычислениях может незаметно исказить результат.

## 🧪 Ответы к задаче 2. Переменная-хамелеон

Исходный код:

magic: int = 42

Последовательные изменения:

magic = "42"  
magic = 42.0  
magic = True

### Что покажет type():

* 42 → <class 'int'>
* "42" → <class 'str'>
* 42.0 → <class 'float'>
* True → <class 'bool'>

### Ответы на вопросы:

* Аннотация int **нарушена**.
* Python не считает это ошибкой, потому что **аннотации — это подсказки**, а не жёсткие ограничения.

## 🧪 Ответы к задаче 3. Обманчивая документация

Исходный код:

user\_age: int = "18"  
average\_score: float = "89.5"  
is\_active: bool = "True"

### Реальные типы:

* user\_age → <class 'str'>
* average\_score → <class 'str'>
* is\_active → <class 'str'>

Все переменные являются строками, несмотря на аннотации.

### Вариант 1. Исправить аннотации

user\_age: str = "18"  
average\_score: str = "89.5"  
is\_active: str = "True"

### Вариант 2. Исправить значения

user\_age: int = 18  
average\_score: float = 89.5  
is\_active: bool = True

### Итоговый вывод

* За соблюдение аннотаций отвечает **программист**, а не Python.
* Аннотации повышают читаемость и надёжность кода, но не заменяют внимательность и дисциплину.