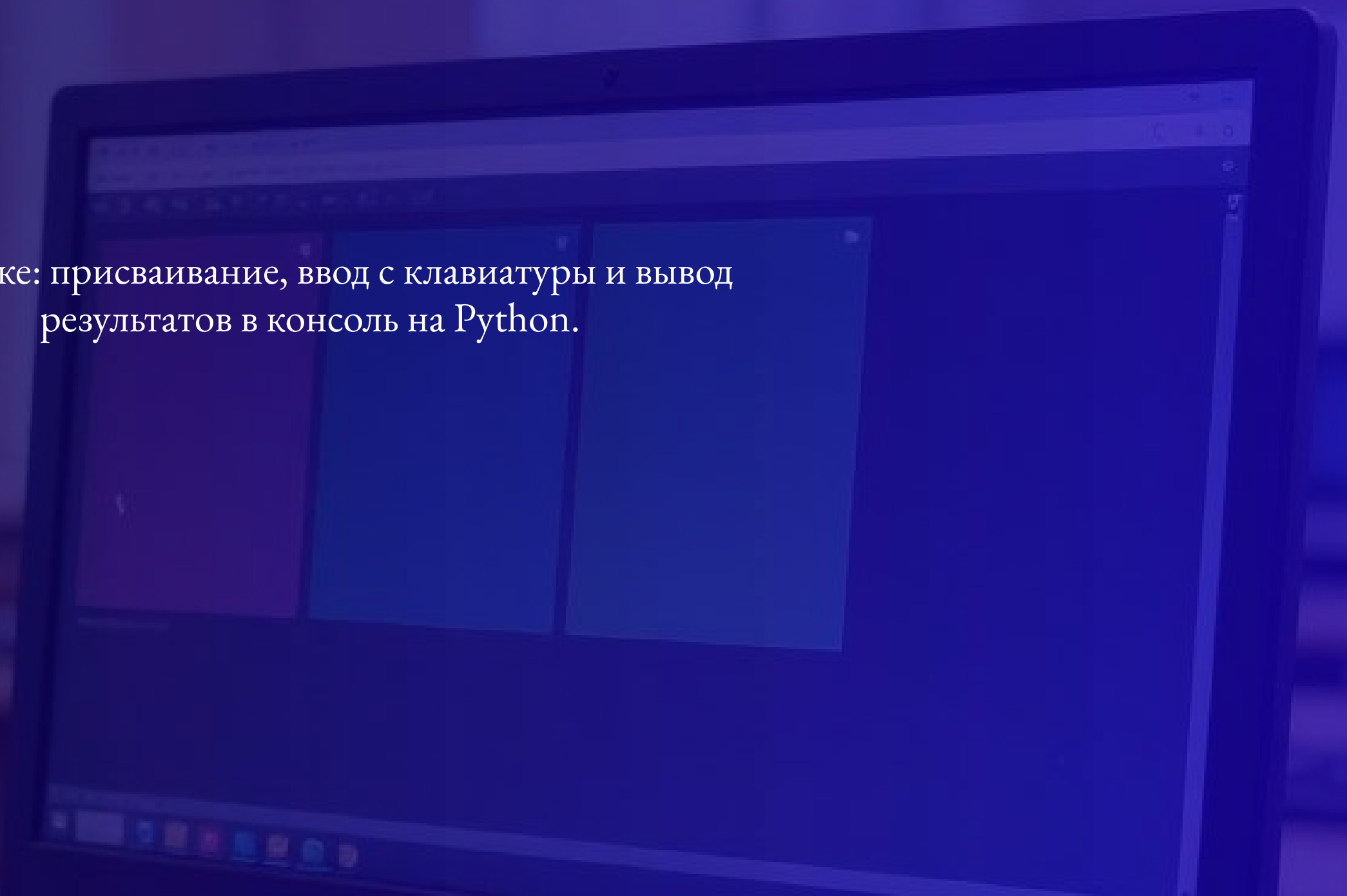


# Оператор присваивания, ввод и вывод данных

На уроке: присваивание, ввод с клавиатуры и вывод  
результатов в консоль на Python.



# Почему без этих трёх действий не работает ни одна программа

Оператор присваивания хранит промежуточные результаты, делая вычисления управляемыми. Ввод делает программу интерактивной, связывая её с пользователем. Вывод показывает итог и служит обратной связью. Вместе они лежат в основе калькуляторов, анкет и тестов.



# Смысл оператора присваивания

В Python присваивание обозначается знаком «=», например `x = 7`. Справа задаётся значение или выражение, которое сначала вычисляется, а затем записывается в переменную слева.

Имя переменной слева «подхватывает» результат справа. При следующем присваивании прежнее значение замещается новым: переменная не накапливает историю, а обновляется.

Присваивание не сравнивает значения. Оно лишь сохраняет результат вычислений, чтобы дальше использовать его в других выражениях и командах программы.

# Присваивание и сравнение: разные операции

| Операция     | Пример               | Смысл               | Результат             |
|--------------|----------------------|---------------------|-----------------------|
| Присваивание | <code>x = 5</code>   | Запись в переменную | x получает 5          |
| Сравнение    | <code>x == 5</code>  | Проверка равенства  | True или False        |
| Путаница     | <code>x = = 5</code> | Неверная запись     | Синтаксическая ошибка |

Сравните, как отличаются «=» и «==»: первое записывает значение, второе проверяет равенство.

Символ «=» работает как сохранение значения, а «==» — как условие сравнения, возвращающее логический ответ.



# Переменная как именованная область памяти

## Что именно хранит переменная

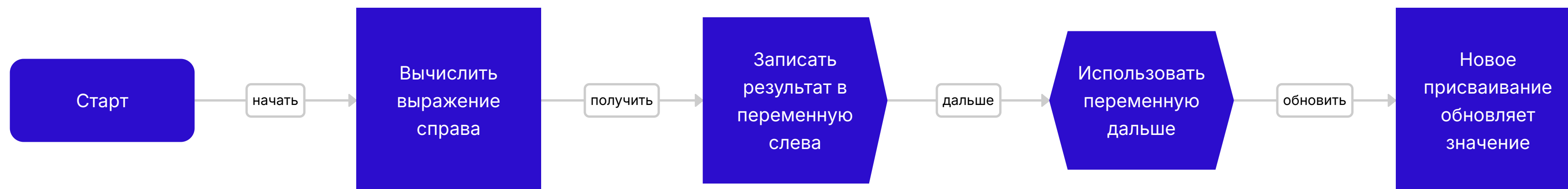
Переменная — это имя, связанное с ячейкой памяти. В каждый момент времени она содержит одно актуальное значение. Когда выполняется новое присваивание, значение обновляется.

## Почему прежние значения исчезают

Старое содержимое не «помнится» автоматически: если оно не сохранено в другой переменной, его уже нельзя использовать. Такая модель помогает понимать связь кода с состоянием памяти.

# Порядок выполнения операции присваивания

Как вычисление справа превращается в значение переменной слева



# Множественное и цепочечное присваивание

Одна строка может присвоить одно и то же значение сразу нескольким переменным:  $a = b = 0$ . Это удобно, когда нужно одновременно инициализировать одинаковые стартовые значения.

Цепочка присваиваний помогает инициализировать счётчики, флаги и временные переменные без повторения выражения. Программа становится компактнее и меньше зависит от ошибок копирования.

Также возможна распаковка:  $x, y = 1, 2$ . Такой приём задаёт сразу два связанных значения, например координаты точки или два промежуточных результата.

# Как меняется значение переменной по шагам программы

После каждого присваивания пересчитывается новое значение, а переменная получает его целиком — предыдущие данные не сохраняются автоматически.

Линейная динамика показывает обновление состояния: переменная всегда содержит последнее присвоенное значение, что важно при последовательных вычислениях.

Схема значений после трёх присваиваний:  $4 \rightarrow 9 \rightarrow 16$

# Ввод данных с клавиатуры

Функция `input()` в Python читает строку из консоли. Даже если пользователь вводит число, для программы это остаётся текстом, то есть значением строкового типа.

Это важно при вычислениях: строку нельзя напрямую складывать с числом. Поэтому часто требуется преобразовать введённый текст в числовой тип, прежде чем выполнять арифметику.

Для числовых значений обычно используют `int()` для целых и `float()` для дробных. Так программа получает корректные типы данных и выполняет вычисления предсказуемо.

# Преобразование ввода в нужный тип данных

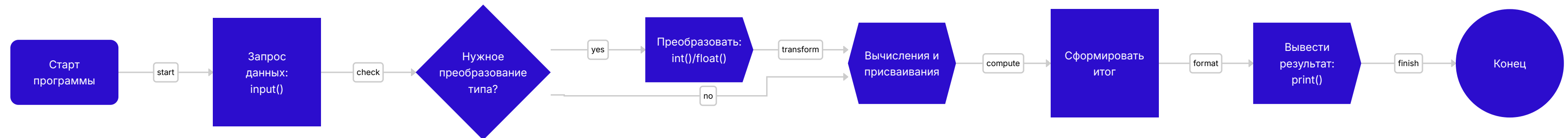
Текст, полученный функцией `input()`, часто нужно привести к нужному типу: целому числу, дробному числу или оставить как строку.

Тип выбирается под формат введённых данных: целые и дробные числа требуют разных преобразований, а некорректная запись приводит к исключению.

| Ситуация           | Пример                                               | Результат              |
|--------------------|------------------------------------------------------|------------------------|
| Целое число        | <code>int(input())</code>                            | Тип <code>int</code>   |
| Дробное число      | <code>float(input())</code>                          | Тип <code>float</code> |
| Текст              | <code>input()</code>                                 | Тип <code>str</code>   |
| Буквы вместо числа | <code>int('abc')</code> или <code>int('12.5')</code> | Ошибка преобразования  |

# Типичный цикл: ввод, обработка, вывод

Старт программы → input() → при необходимости преобразование типа → вычисления и присваивания → формирование результата → print() → завершение.



# Функция вывода print()

- Функция print() отображает данные в консоли. С её помощью печатают текстовые сообщения, значения переменных и результаты вычислений: программа становится «видимой» для ученика.
- Когда передают несколько аргументов, print() разделяет их пробелами. В практических заданиях вывод часто включает приглашение к вводу, ответ пользователя и итог вычислений.



# Что чаще всего выводит программа

В учебных примерах вывод обычно не ограничивается «числом»: добавляют пояснения, чтобы результат было легко интерпретировать и сравнить с ожиданиями.

Диаграмма показывает, что значимая доля вывода — это организация диалога: человек вводит данные, а программа комментирует шаги и финальный ответ.

# Частые ошибки при работе с вводом и выводом

Путаница между `=` и `==` часто приводит к неправильной логике или синтаксическим сбоям. Если перепутать операторы, условие перестает проверять равенство и начинает перезаписывать данные.

Вторая частая проблема — арифметика над строкой. Поскольку `input()` возвращает всегда строку, выражения вроде

Ещё одна ловушка связана с оформлением результата: забытые пояснения, лишние пробелы и неаккуратный формат строки затрудняют чтение вывода и усложняют проверку ответов.

При формировании строк легко неверно расставить запятые в `print()` и получить неожиданное количество аргументов. Чтобы избежать этого, вывод делают структурированным и проверяют формат на нескольких примерах.

# Типичные ошибки и их последствия

| Ошибка               | Пример                       | Последствие                                                                           |
|----------------------|------------------------------|---------------------------------------------------------------------------------------|
| Синтаксис            | <code>x = = 5</code>         | Программа не запускается: синтаксическая ошибка.                                      |
| Типы данных          | <code>x = input() + 1</code> | Ошибка типов: строка нельзя сложить с числом.                                         |
| Преобразование       | <code>int('12.5')</code>     | Ошибка преобразования: нельзя получить <code>int</code> из числа с десятичной точкой. |
| <code>print()</code> | <code>print(2, 3)</code>     | Вывод двух значений через пробел в одной строке.                                      |

Столбцы: ошибка, пример и последствие. Разбор показывает, что большинство проблем рождаются из синтаксиса и несоответствия типов данных при вводе.

Корректность программы зависит от двух проверок: правильного синтаксиса и согласованности типов. Если типы не совпадают, даже логичная идея не выполнится.

# Практический порядок действий в простой программе

## 1. Запрос данных

Программа обращается к пользователю и получает ввод через `input()`. Этот шаг задаёт «сырьё» для дальнейших вычислений: без него обработка невозможна.

## 2. Сохранение в переменных

Полученные значения записывают в переменные, чтобы использовать снова и снова. Так данные не теряются между строками и проще контролировать логику алгоритма.

## 3. Обработка и вычисления

Далее выполняют вычисления и при необходимости обновляют промежуточные результаты новыми присваиваниями. Вычисления опираются на уже сохранённые переменные.

## 4. Формирование ответа и вывод

В конце формируют итоговую строку или число и выводят его через `print()`. Пользователь видит результат, а формат вывода помогает понять, что именно произошло.

# Состав простого диалога программы с пользователем

Даже короткая программа обычно напоминает диалог: сначала звучит приглашение, затем ввод попадает в переменные, после чего происходит вычисление и отображается результат.

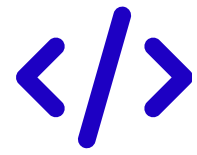
Главная «масса» логики приходится на обработку данных. Понимание распределения этапов помогает проектировать программы, которые не путают пользователя.



# Мини-программа с вводом, присваиванием и выводом

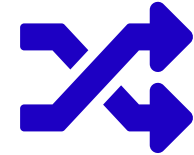
1. Мини-пример начинается с ввода имени через `input()`. Затем возраст запрашивают второй строкой и преобразуют в число с помощью `int(input())`. После этого значения сохраняют в переменных для дальнейшего использования.
2. На следующем этапе программа формирует сообщение, опираясь на сохранённые данные, и печатает его через `print()`. Пользователь видит осмысленный ответ: как введённые значения превратились в результат.

# Связь темы с дальнейшим изучением программирования



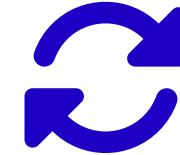
## Основа логики программы

Присваивание связывает данные с именами, а ввод и вывод делают программу интерактивной. Это фундамент, на котором строятся следующие темы: условия, циклы и работа с результатами.



## Проверки условий

Когда вы сравниваете значения (например, возраст) и используете `if`, вам нужны корректные переменные. Ошибки ввода и типов данных сразу отражаются на логике ветвлений.



## Циклы повторения действий

Циклы часто накапливают результат через последовательные присваивания. Умение печатать промежуточные значения через `print()` помогает отлаживать такие алгоритмы.



## Функции и обработка данных

Функции принимают параметры, которые получают из `input()`, и возвращают результаты. Устойчивые правила преобразования типов и понятный вывод делают вычисления надёжными.

# Ключ к первым работающим программам

Оператор присваивания сохраняет значения в переменных и обновляет данные в памяти программы. `input()` получает информацию от пользователя, а `print()` делает результат видимым на экране. Различайте `=` и `==` и учитывайте типы данных, чтобы избегать ошибок.