

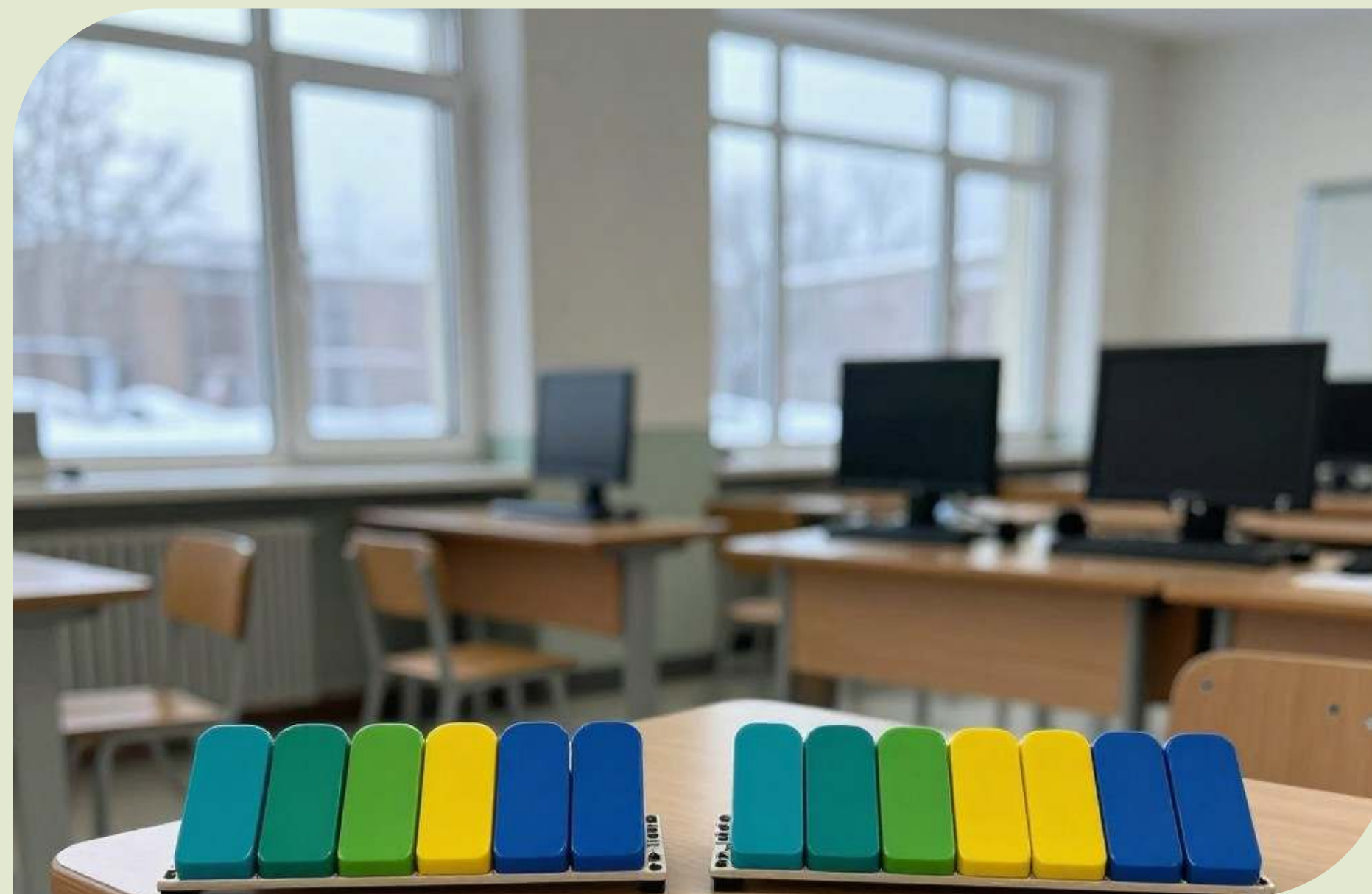


Кодирование основных типов алгоритмических структур

Как записывать следование, ветвление и циклы в
процедурном и объектно-ориентированном коде: Python
и C++.

Один алгоритм — разные записи

Алгоритм — это точная последовательность действий для решения задачи. В школе его оформляют словами, блок-схемами, псевдокодом и программным кодом. Процедурный подход описывает шаги и функции, ООП объединяет данные и действия в объектах. Логика одна, форма записи разная.



Три базовые алгоритмические структуры

01

Следование: команды выполняются одна за другой в заданном порядке. Если шагов несколько, важна последовательность — «сначала ввод, потом вычисление, затем вывод». Так работает большинство учебных расчетов.

02

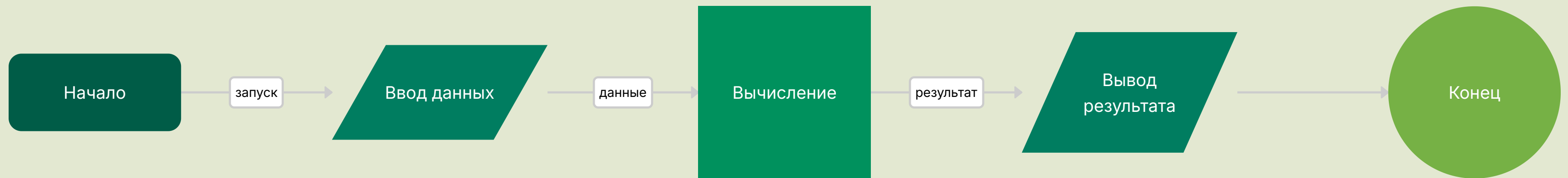
Ветвление: выполнение зависит от условия. Программа проверяет факт и выбирает ветку — например, ответ «да» или «нет», положительное или отрицательное число, чётное или нечётное значение.

03

Цикл: группу команд повторяют несколько раз. Повторение нужно, когда действий много или данные приходят списком. Часто структуры комбинируются: внутри цикла бывает ветвление, а после следования — новое повторение.

Структура следования в блок-схеме и коде

Линейный путь: начало → ввод → вычисление → вывод → конец



Ветвление: выбор по условию

Простая проверка

Ветвление выполняет одну ветку из нескольких, опираясь на результат проверки. В Python это обычно записи `if`, `else`, `elif`, а в C++ — `if`, `else if`, `else`. Так программа выбирает действие при разных входных данных.



Условие с несколькими вариантами

Когда вариантов больше, форма записи меняется: сначала идёт проверка, затем — возможные альтернативы. Это удобно для задач с диапазонами и правилами: деление на ноль, выбор оценки по баллам, определение знака числа и проверка возраста.

Когда нужен if, а когда несколько вариантов выбора

Если условий проверяется один раз, обычно хватает простого if. При двух исходах удобнее if else. Для множества вариантов применяют elif или case-подобные конструкции.

Набор учебных примеров (проверка баллов, день недели по номеру, выбор команды)

1 вариант 2 варианта Много вариантов



Диаграмма показывает: в школьных заданиях чаще всего встречаются простые шаблоны выбора — один, два или несколько фиксированных вариантов.

Циклы: повторение действий без лишнего кода



for — когда число повторов известно

Цикл for используют, когда заранее понятно, сколько раз нужно повторить действия или по какому диапазону вести перебор значений, например для вывода 1–10 или суммы по списку.



while — пока условие истинно

Цикл while подходит, когда нужно продолжать работу до наступления условия: например, повторять ввод пароля, пока пароль не окажется правильным, или собирать сумму до нужного предела.



Экономия кода и меньше ошибок

Циклы помогают не дублировать строки программы вручную. Это особенно важно при обработке нескольких значений: таблица умножения, подсчёты по списку, перебор оценок.

Одна задача — разные стили программирования

Задача	Процедурный код	Объектно-ориентированный код
Сумма чисел	функция суммы	объект с методом расчёта
Проверка условия	функция проверки	метод объекта-«проверки»
Повторение действий	цикл в функции	метод объекта + внутренний цикл
Несколько похожих объектов	отдельные вызовы функций	класс, который создаёт много экземпляров

Ключевое отличие не в логике, а в организации: процедурный стиль упорядочивает шаги функциями, ООП связывает данные и поведение в классе.

Ниже — сравнение того, как одна и та же логика может быть организована по-разному: через функции и вызовы или через классы и методы.

Учебные примеры на Python и C++

Процедурное программирование: функции и шаги решения

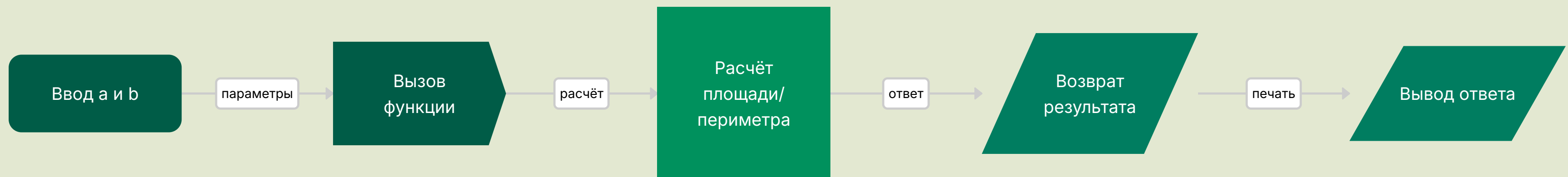


● В процедурном стиле программу удобно разбивать на функции: каждая решает маленькую задачу. Данные передаются через параметры, а результат возвращается значением функции. Такой подход помогает держать порядок действий.

● Например, отдельно могут быть реализованы функция вычисления площади, проверка числа, поиск максимума или обработка ввода. Компоненты затем соединяются последовательностью вызовов, образуя понятный алгоритм.

Процедурная схема решения простой задачи

Пример: вычислить площадь или периметр прямоугольника по двум сторонам



ООП: класс, объект, метод

1.

Класс задаёт общий шаблон для объектов: какие данные они хранят и какие действия выполняют. Это «чертёж», а не конкретика, поэтому по одному классу можно создавать много разных экземпляров.

2.

Объект — это конкретный экземпляр класса с собственными значениями свойств. Например, ученик хранит имя и класс, а книга — название и автора; счёт игры хранит очки и состояние команд.

3.

Метод — действие, которое объект умеет выполнять. Он использует данные объекта и меняет его состояние, например увеличить очки, сбросить счёт или получить текущее значение для вывода на экран.

Инкапсуляция и защита данных объекта



● Инкапсуляция означает, что внутренние данные объекта меняются через его методы. Снаружи код обращается к понятным действиям, а «как именно всё устроено внутри», остаётся скрытым, уменьшая шанс ошибок.

● В объекте «Счётчик» значение хранится внутри, а снаружи вызываются методы: увеличить, уменьшить, изменить или получить значение. Такое разделение напоминает коробку с замком: часть информации защищена от прямого вмешательства.

Где чаще удобен процедурный подход, а где ООП

Большинство школьных задач можно решать по-разному, но структура решения подсказывает стиль: когда нужна «цепочка шагов» — удобнее процедурный код, когда важны «объекты и их действия» — ООП.

Обобщённые учебные рекомендации по стилям записи алгоритмов (без конкретных числовых измерений), ориентировано на задания 9 класса.

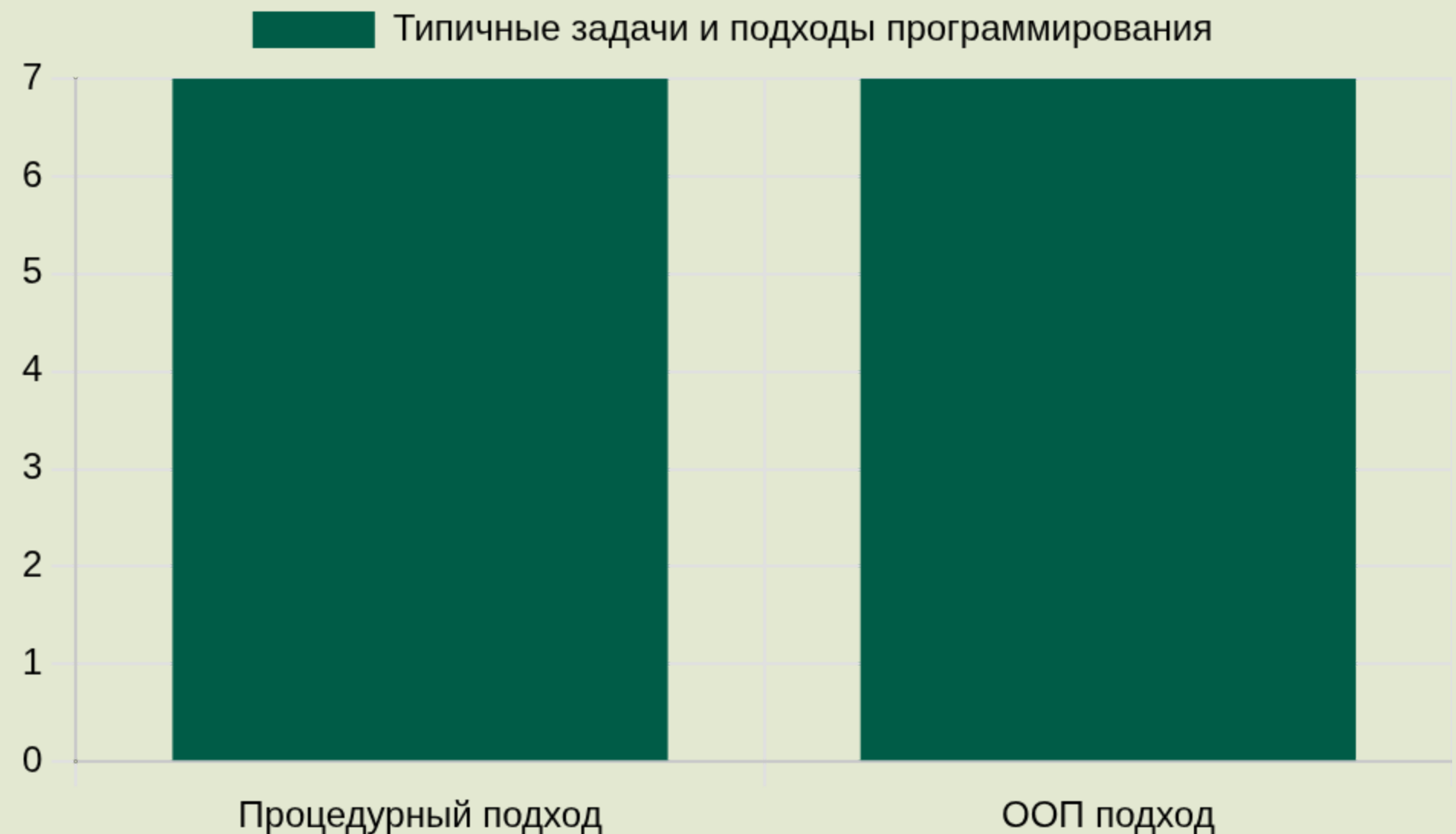


Диаграмма показывает идею выбора: оба подхода применимы, но ООП чаще выигрывает при повторном использовании похожих объектов и при расширении проекта новыми сущностями и поведением.

Ветвление на Python и C++: одинаковый смысл, разный синтаксис

Сравните, как записывается выбор ветки: в Python используются отступы и двоеточие, в C++ — фигурные скобки и точка с запятой. Смысл условия сохраняется.

Сопоставление синтаксиса конструкций ветвления в школьных курсах Python и C++ (обобщённый учебный материал).

Ситуация	Python	C++
Чётность числа	<code>if n % 2 == 0:</code>	<code>if (n % 2 == 0) {</code>
Сравнение чисел	<code>if a > b: elif a < b:</code>	<code>if (a > b) {} else if (a < b) {}</code>
Оценка по диапазону	<code>if-elif по границам</code>	<code>if-else if по границам</code>

Разный внешний вид записи не меняет логики: сначала проверяется условие, затем выбирается нужная ветка выполнения. Учащимся важно читать условие, а не запоминать форму скобок.

Цикл for в Python и C++

01

В Python цикл for часто перебирает элементы последовательности или значения, полученные из range. Это удобно, когда заранее понятно, какие и сколько значений нужно обработать.

02

В C++ конструкция for обычно содержит счётчик, условие продолжения и шаг изменения счётчика. Такая форма помогает явно видеть, как именно меняется перебор на каждом шаге.

Оба варианта подходят для повторения заранее известного числа раз: вывод чисел от 1 до 10, подсчёт суммы, печать таблицы умножения и обработка списка оценок. В обоих случаях повторение — ключевой смысл.

03

Различается лишь запись: где в одном языке перечисление или range, там в другом языке управляют счётчиком и условием. Благодаря этому легко переносить одну идею повторения между языками.

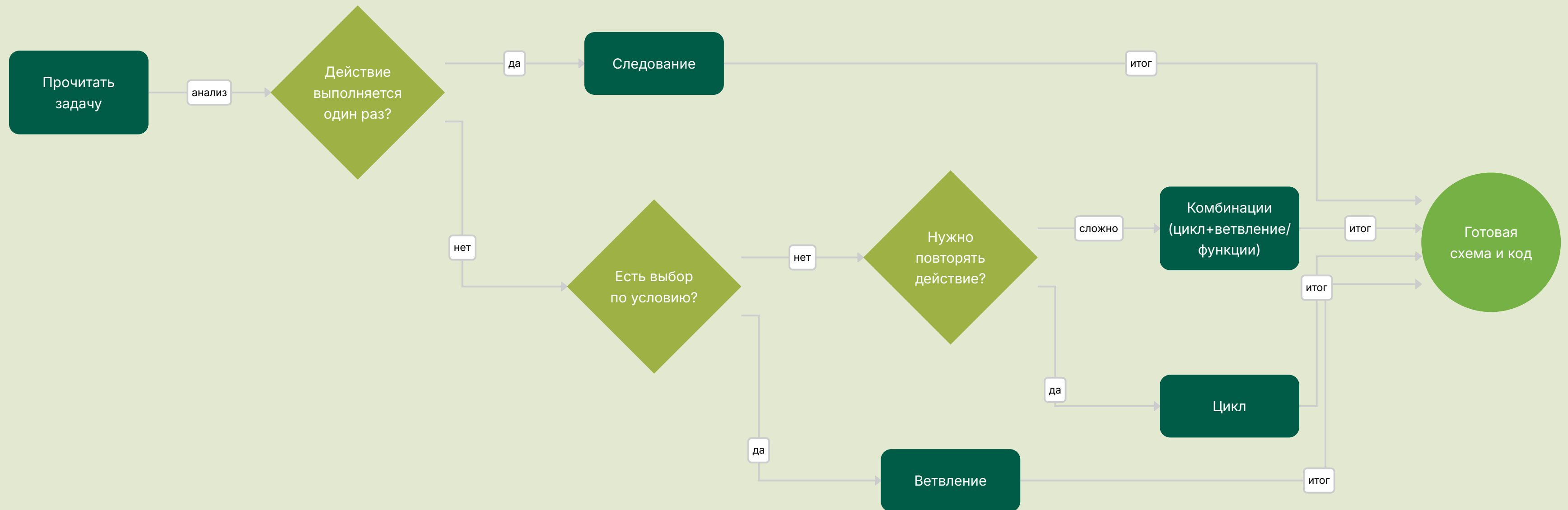
04

Цикл `while` и защита от бесконечного повторения



Выбор структуры алгоритма

Логика подбора: один раз — следование, выбор — ветвление, повтор — цикл; затем структуры можно комбинировать.



Типичные ошибки при записи алгоритмов

Большинство проблем появляется не из-за вычислений, а из-за логики: условие может не сработать, цикл может не завершиться, а порядок шагов может нарушить смысл задачи.

Обобщённые наблюдения по типичным ошибкам учеников в задачах на алгоритмические структуры (учебная практика).



Если внимательно проверять условия, индексы/счётчики, отступы и последовательность команд, можно резко уменьшить число ошибок. Это особенно важно в циклах и ветвлениях.

Учебные задачи для закрепления темы



Сумма чисел

Решите задачу на подсчёт суммы: сначала следование задаёт порядок шагов, затем при необходимости добавьте повторение, чтобы обработать несколько чисел подряд.



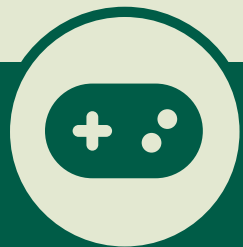
Проверка чётности

Выберите ветвление по условию: если остаток от деления на 2 равен нулю — число чётное, иначе — нечётное. Так тренируется if и логические проверки.



Таблица умножения

Составьте программу, которая выводит таблицу умножения. Для повторения удобны циклы, а правильный порядок действий связывает ввод, вычисление и вывод результата.



Игра с очками

Сделайте мини-скрипт: очки увеличиваются или сбрасываются по командам. Здесь полезны ветвления и работа со значением состояния как с переменной.



Оценки по баллам

Напишите обработку результатов: по диапазонам баллов выбирается итоговая оценка. Задача закрепляет множественный выбор через цепочку условий или эквивалентные конструкции.



Главный вывод: важна логика алгоритма и точность записи

Одна и та же задача может быть представлена в процедурном или объектно-ориентированном стиле, но базовые структуры остаются: следование, ветвление и цикл. Тогда проще читать код, избегать логических ошибок и переносить решение между языками.

