МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

ОЛИМПИАДА «Я-БАКАЛАВР» ДЛЯ ОБУЧАЮЩИХСЯ 5-11 КЛАССОВ

ИНФОРМАТИКА

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ ПОДГОТОВКИ

К ЗАКЛЮЧИТЕЛЬНОМУ ЭТАПУ ОЛИМПИАДЫ

2025/2026 УЧЕБНОГО ГОДА ДЛЯ **11** КЛАССА

ЗАКЛЮЧИТЕЛЬНЫЙ ЭТАП

Характер и уровень сложности олимпиадных задач направлены на достижение целей проведения олимпиады: выявить способных участников, твердо владеющих школьной программой и наиболее подготовленных к освоению образовательных программ ВУЗов, обладающих логикой и творческим характером мышления знанием вопросов информатики, вычислительной техники, информационных и коммуникационных технологий.

Задания дифференцированы по сложности и требуют различных временных затрат на верное и полное решение. Задания направлены на выявление интеллектуального потенциала, аналитических способностей и креативности мышления участников и т.п.

Очный этап олимпиады проводится только в письменной форме. Каждый участник олимпиады получает бланк с заданием, содержащий 6 заданий. При выполнении заданий требуется:

1. владеть навыками составления графических алгоритмов для решения любых задач;
2. владеть навыками расчётов логических выражений;
3. владеть навыками простого шифрования;
4. владеть навыками программирования.

При подготовке к олимпиаде следует повторить приведенные ниже темы.

ПЕРЕЧЕНЬ ЭЛЕМЕНТОВ СОДЕРЖАНИЯ, ВКЛЮЧЕННЫХ В ЗАДАНИЯ ОЛИМПИАДЫ ЗАКЛЮЧИТЕЛЬНОГО ЭТАПА 2025/2026 УЧЕБНОГО ГОДА

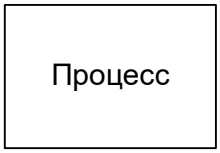
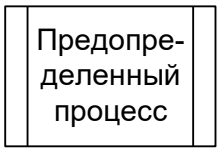
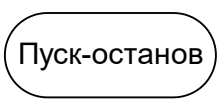
Тема 1. Составление графического алгоритма для решения задач

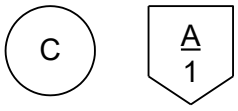
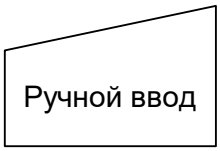
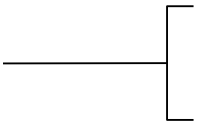
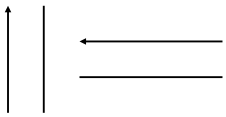
Графический способ описания алгоритмов представляет собой изображение структуры алгоритма, при котором все этапы процесса обработки данных представляются с помощью определенного набора геометрических фигур – блоков, имеющих строгую конфигурацию в соответствии с характером выполняемых действий. Такое графическое представление алгоритма называется схемой алгоритма. Составление алгоритмов осуществляется в соответствии с требованиями ГОСТ 19701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения» и ГОСТ 19.003-80 «Схемы алгоритмов и программ. Обозначения условные графические». По назначению и характеру отображаемых функций условные графические изображения делятся на основные и вспомогательные. Основные символы используются для представления операций, раскрывающих характер обработки данных в процессе решения задачи. Вспомогательные символы предназначены для

пояснения отдельных элементов схемы алгоритма. Изображение схем алгоритмов осуществляется по определенным правилам. Каждая схема должна начинаться и заканчиваться символами, обозначающими начало и окончание алгоритма. Все блоки в схеме располагаются в последовательности сверху вниз и слева направо. Отдельные блоки соединяются между собой линиями потоков информации. Направление линии потока сверху вниз и справа налево принимаются за основные и стрелками не обозначаются в отличие от других направлений. Необходимым условием является один выход из символов, обозначающих обрабатывающие блоки, и двух выходов из символов, обозначающих логические операции по проверке выполнения условий.

Приведем основные условные обозначения функциональных блоков схем алгоритмов (Таблица 1).

Таблица 1 – Обозначения, название и функциональное назначение блоков алгоритмов

Обозначение	Название и функциональное назначение
	Процесс – вычислительная операция или группа операций
	Решение – определяет выбор направления выполнения алгоритма в зависимости от условия, записанного внутри блока
	Модификация – заголовок цикла
	Предопределенный процесс – использование ранее разработанного алгоритма как составной части решения задачи, подпрограмма
	Пуск-останов – начало, конец алгоритма, вход в подпрограмму, выход из подпрограммы

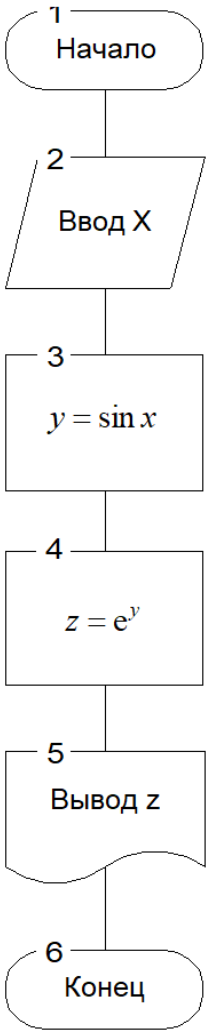
Обозначение	Название и функциональное назначение
	Соединитель и межстраничный соединитель. Указывают связь между прерванными линиями
	Ввод-вывод – обобщенный блок ввода вывода
	Ручной ввод – ввод данных с клавиатуры
	Дисплей -вывод информации на экран дисплея
	Документ - вывод, печать информации на бумажный носитель
	Комментарий - пояснения к операции блока
	Линии потока - изображение связи между блоками. Линии без стрелок указывают направление потока слева направо или сверху вниз

Функциональным блокам схемы алгоритма могут присваиваться порядковые номера, которые проставляются слева в верхней части символов в разрыве их контура.

Алгоритм может быть записан на одном из языков программирования. Под языком программирования понимается формальный язык, воспринимаемый ЭВМ и предназначенный для общения человека с машиной. Алгоритм, записанный на языке программирования, называется программой.

В этом случае алгоритм представляется в виде последовательности операторов языка.

Пример задания: определим величину $z = e^y$ при $y = \sin x$. Линейный вычислительный процесс и структура следование представлен на рисунке ниже.



Тема 2. Логические основы компьютера

Логические операции могут производиться не только над булевыми величинами, но и над битами операндов. В этом случае логическая операция возвращает поразрядный результат, который либо истинен (1), либо ложен (0). В языках программирования могут существовать специальные операторы побитового выполнения логических операций. Например, в «Си++» и «Ява» поразрядным (побитовым) операциям НЕ, И, ИЛИ соответствуют операторы ~, &, | (сравните с операторами таблицы 7).

В Бейсике используются только побитовые логические операции, а операнды представляются в восьми-, шестнадцати- или тридцатидвухразрядном дополнительном коде. При этом булевым значениям

False и True соответствуют десятичные значения 0 и -1, так 0 — число, в котором все биты обнулены, а -1 — двоичное число, все биты которого установлены в 1 (таблица 3).

Операциям *исключающее ИЛИ* (неравнозначность), *эквивалентность* (равнозначность) и *импликация* в Бейсике соответствуют операторы XOR, EQV и IMP. Результат логической операции определяется поразрядно согласно таблице 8. Операторы приведены в порядке убывания их приоритета.

Таблица 8 - Результаты, возвращаемые логическими операциями

Операнды		Результаты операций					
X	Y	NOT X	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
1	1	0	1	1	0	1	1
1	0	0	0	1	1	0	0
0	1	1	0	1	1	0	1
0	0	1	0	0	0	1	1

Пример задания:

Воспользуемся приемами для упрощения записи сформулированного выражения из алгебры логики. В процессе преобразований вынесем ¬ у выражения А за знаки скобок. Далее целесообразно воспользоваться закономерностью исключенного третьего и распределительным способом. В итоге вычислений получим следующее равенство. Произведем необходимые алгебраические операции в уравнении и запишем результирующий ответ:

$$F = \overline{A}\,\&\,\overline{B} \vee \overline{A}\,\&\,B \vee A\,\&\,B = \overline{A}\,\&\,(\overline{B} \vee B) \vee A\,\&\,B = \overline{A} \vee A\,\&\,B = (\overline{A} \vee A)\,\&\,(\overline{A} \vee B) = \overline{A} \vee B$$

Ответ: ¬A∨B

Пример задания:

Вычислить логическое выражение.

Y = (38 OR &H1C) AND &H15 IMP NOT &O5

Y1=1 Y2=10100 Y3=1111111111111011 Y4=111110

Р а с ч е т з а д а н и я

Переводим все операнды в двоичную систему счисления:

$38_{(10)} = 100110_{(2)}$

$1C_{(16)} = 11100_{(2)}$

$15_{(16)} = 10101_{(2)}$

$5_{(8)} = 101_{(2)}$

Указываем приоритет выполнения операций:

Y = (38 OR &H1C) AND &H15 IMP NOT &O5

1 2

3 4

Определяем результат выполнения каждой операции побитно, используя для представления операндов шестнадцатиразрядный дополнительный код:

1) 38 **OR** &H1C

```
  0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0
  0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0
-------------------------------
  0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0
```

2) **NOT** &O5

```
  0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
-------------------------------
  1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0
```

3) (38 OR &H1C) **AND** &H15

```
  0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0
  0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1
-------------------------------
  0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
```

4) (38 OR &H1C) AND &H15 **IMP** NOT &O5

```
  0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
  1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0
-------------------------------
  1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
```

Ответ: Y = Y3 = $1111111111111011_{(2)}$ = $177773_{(8)}$ = $FFFB_{(16)}$ = $-5_{(10)}$

Тема 3. Простые алгоритмы шифрования

Существует много различных систем шифрования. К ним прибегают в военном деле, на дипломатической службе и в других случаях, когда нужно сохранить в тайне содержание переписки. Шифрование текста используется человечеством с того самого момента, когда появилась первая секретная информация – такая, которая должна быть недоступна тем, кому она не предназначена.

Шифр Цезаря один из самых первых известных методов шифрования носит имя римского императора Юлия Цезаря (I в. до н. э.), который если и не изобрел его сам, то активно им пользовался. Этот метод основан на замене каждой буквы шифруемого текста на другую путем смещения в алфавите от исходной буквы на фиксированное количество символов; причем алфавит читается по кругу, т.е. после буквы «я» снова идет буква «а». Регистр символов при этом не учитывается.

Пример задания:
Необходимо закодировать слово «БАЙТ».

Кодировочная таблица:

А	Б	В	Г	Д	Е	Ё	Ж	З	И	Й
01	02	03	04	05	06	07	08	09	10	11
К	Л	М	Н	О	П	Р	С	Т	У	Ф
12	13	14	15	16	17	18	19	20	21	22
Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
23	24	25	26	27	28	29	30	31	32	33

Используя кодовую таблицу выполняем смещение каждого символа на два символа вправо. Таким образом, получаем кодовое слово «ГВЛФ».

Ответ. «ГВЛФ»

Тема 4. Программирование

Программирование – это процесс и искусство создания компьютерных программ с помощью языков программирования. В более широком смысле оно представляет собой разработку программного обеспечения. Программирование сочетает в себе элементы искусства, науки, математики и инженерии и подразумевает написание исходного кода, который потом компилируется или интерпретируется в машинный код для выполнения на компьютере.

Основная задача программирования — это реализация алгоритмов, то есть последовательностей действий, которые компьютер должен выполнить для решения конкретной задачи. При этом программисты выбирают подходящий язык программирования, учитывая тип задачи и требования к производительности и удобству разработки.

Основные понятия в программировании включают переменные (хранение данных), инструкции (команды для компьютера) и выражения (вычисления). Программирование широко используется в самых разных сферах: от разработки сайтов и приложений до автоматизации бизнес-процессов, медицины и образования

Каждый язык имеет свои собственные особенности и тонкости реализации, которые можно узнать на официальных страницах этих языков.

Например:

Python https://www.python.org/

Java https://www.java.com/ru/

C# https://dotnet.microsoft.com/ru-ru/languages/csharp

C++ https://learn.microsoft.com/ru-ru/cpp/?view=msvc-170

Эти ссылки являются основными для получения теоретических знаний о построении и особенностях применения конкретного языка программирования.

Для подготовки к заключительному этапу необходимо повторить:

Общие вопросы программирования.

Для успешного освоения программирования необходимо изучить широкий спектр общих вопросов. Начните с основ. Изучите, что такое алгоритм, как его представить в виде блок-схемы или псевдокода, и освойте основные алгоритмические структуры, такие как последовательность, ветвление и циклы, а также базовые алгоритмы поиска и сортировки.

Углубитесь в понимание типов данных, включая основные (целые числа, числа с плавающей точкой, символы, строки, логические значения) и структуры данных (массивы, связные списки, стеки, очереди, деревья, графы, хэш-таблицы). Необходимо понимать, как объявлять и инициализировать переменные и константы, а также учитывать область их видимости. Освойте арифметические, логические операторы, операторы сравнения и присваивания.

Изучите парадигмы программирования, такие как императивное, декларативное, структурное, объектно-ориентированное и функциональное программирование, а также принципы SOLID. Необходимо понимать структуры управления программой, включая условные операторы, циклы, функции, процедуры и обработку исключений.

Необходимо ориентироваться в терминах, что такое алгоритм, компилятор, интерпретатор, синтаксис, исходный код, парадигма программирования, отладка программы, API, IDE, и т.д. Знать основы работы с программным кодом.

Типы и структуры данных.

В программировании типы и структуры данных играют ключевую роль в организации и эффективной обработке информации. Фундаментом являются примитивные типы данных, такие как целые числа (int, short, long, byte), числа с плавающей точкой (float, double), символы (char), строки (String) и логические значения (boolean). Важно понимать их особенности и диапазоны значений.

Составные типы, такие как массивы и структуры, строятся на основе примитивных типов, позволяя группировать связанные данные.

Структуры данных, в свою очередь, предоставляют различные способы организации и хранения, влияя на производительность программы.

Линейные структуры включают связные списки, стеки, очереди и деки, каждая из которых имеет свои особенности и принципы работы (LIFO, FIFO).

Нелинейные структуры, такие как деревья и графы, предоставляют иерархические и сетевые способы организации данных.

Ассоциативные структуры, такие как хэш-таблицы, словари и множества, обеспечивают быстрый поиск и хранение уникальных элементов.

Абстрактные типы данных (ADT) определяют что структура данных делает, а не как. Список, стек, очередь, дерево и граф являются примерами ADT.

Выбор подходящей структуры данных зависит от требований к производительности, объема данных, типов операций и простоты реализации.

При изучении необходимо понимать концепции указателей и ссылок, динамического выделения памяти, рекурсии, обобщенного программирования и сложности алгоритмов. Рекомендуется изучать теорию, практиковаться в реализации структур данных, рассматривать примеры их использования и анализировать эффективность различных структур для конкретных задач.

Условные конструкции.

Разберитесь с базовым синтаксисом работы условных конструкций. Для этого рассмотрите операторы:

if (если): Самый простой условный оператор. Выполняет блок кода только в том случае, если условие истинно.

```
if условие:
    # Код, который выполняется, если условие истинно
```

else (иначе): Выполняет блок кода, если условие if ложно.

```
if условие:
    # Код, который выполняется, если условие истинно
else:
    # Код, который выполняется, если условие ложно
```

elif (иначе если) / else if: Позволяет проверять несколько условий последовательно. Выполняется блок кода, соответствующий первому истинному условию.

```
if условие1:
    # Код, если условие1 истинно
elif условие2:
    # Код, если условие2 истинно
else:
    # Код, если ни одно из условий не истинно
```

Условия (логические выражения).

Операторы сравнения:

== (равно)

!= (не равно)

> (больше)

< (меньше)

>= (больше или равно)

<= (меньше или равно)

Логические операторы:

and / && (логическое И). Условие истинно, если оба операнда истинны.

or / || (логическое ИЛИ). Условие истинно, если хотя бы один операнд истинен.

not / ! (логическое НЕ). Инвертирует значение операнда (истина становится ложью, и наоборот).

Оператор **switch** (в некоторых языках):

Альтернатива цепочке **if-elif-else** для проверки равенства переменной с несколькими конкретными значениями.

Пример для c++

```
switch (переменная) {
   case значение1:
      // Код для значения1
      break;
   case значение2:
      // Код для значения2
      break;
   default:
      // Код, если ни одно из значений не совпало
}
```

Важно помнить про break, чтобы избежать "проваливания" в следующие case.

Массивы.

Массив – это упорядоченная структура данных, содержащая элементы одного типа, которые расположены в памяти последовательно. Каждый элемент доступен по своему индексу, начиная, как правило, с нуля.

Массивы бывают статические, размер которых фиксируется во время компиляции и не может быть изменен, и динамические, размер которых можно изменять во время выполнения программы. Также существуют многомерные массивы, представляющие собой массивы массивов и используемые для представления, например, таблиц.

Основные операции над массивами включают создание, доступ к элементам по индексу (чтение и запись), итерацию (перебор всех элементов),

изменение размера (для динамических массивов), поиск элементов и сортировку. Важно помнить об ошибке выхода за границы массива.

Существуют различные алгоритмы для работы с массивами, такие как линейный и бинарный поиск, а также алгоритмы сортировки: пузырьковая сортировка, сортировка вставками, выбором, быстрая сортировка и сортировка слиянием. Важно понимать, как сложность алгоритмов (оценка их эффективности) влияет на время выполнения операций с массивами, особенно с большими объемами данных.

В разных языках программирования массивы имеют свои особенности. В C/C++ важен контроль типов и ручное управление памятью (или использование умных указателей), в Java массивы являются объектами, а в Python списки предоставляют удобные встроенные методы для работы с данными.

В Python и JavaScript поддерживаются динамические массивы с гибкой типизацией и автоматическим управлением памятью.

Для изучения массивов рекомендуется начать с теории, затем перейти к практике решения задач, использовать отладчик для понимания работы кода и анализировать существующие примеры кода.

Циклы.

Цикл в программировании – это конструкция, позволяющая многократно выполнять определенный блок кода, который называется телом цикла. Повторение происходит до тех пор, пока выполняется заданное условие цикла.

В циклах часто используется счетчик (итератор) – переменная, которая изменяется на каждой итерации и помогает контролировать выполнение цикла.

Существуют три основных типа циклов:

— for (цикл со счетчиком);

— while (цикл "пока");

— do-while (цикл "делай-пока").

Синтаксис циклов немного отличается в разных языках программирования, но основная логика остается одинаковой.

Управление циклом осуществляется с помощью операторов break (немедленный выход из цикла) и continue (пропуск текущей итерации). Важно избегать создания бесконечных циклов, которые могут возникнуть, если неверно задано условие выхода.

Вложенные циклы – это циклы, помещенные внутри других циклов, которые полезны для обработки многомерных массивов и других сложных структур данных.

Циклы находят широкое применение в программировании, например, для обработки массивов, чтения данных из файлов, создания графических

интерфейсов и реализации различных алгоритмов. Важно уметь оптимизировать циклы, уменьшая количество итераций и вынося инвариантный код за их пределы.

При работе с циклами часто возникают ошибки, такие как бесконечные циклы, смещение индекса, неправильная инициализация счетчика и неверное использование break и continue.

Использование отладчика и анализ существующего кода помогают избежать этих ошибок и лучше понять работу циклов. В некоторых языках существуют продвинутые концепции, такие как итераторы и генераторы в Python, которые предоставляют более эффективные способы перебора последовательностей. Также бывают доступны циклы foreach и параллельные циклы для упрощения работы с коллекциями и повышения производительности.

Пример задания. «Анализатор температурного журнала».

Необходимо разработать программу, которая обрабатывает данные о ежедневной температуре воздуха в течение месяца, выполняет базовый статистический анализ и предоставляет результаты.

Входные данные:

Массив (список) из 30 целых чисел, представляющих ежедневную температуру воздуха за месяц в градусах Цельсия. Температуры вводятся вручную.

Требуется выполнить:

Вычислить и вывести среднюю температуру за месяц.

Подсчитать и вывести количество дней в месяце, когда температура была ниже 0 градусов Цельсия.

Найти и вывести максимальную температуру за месяц и день (индекс в массиве + 1), когда она была зафиксирована впервые.

Найти и вывести минимальную температуру за месяц и день (индекс в массиве + 1), когда она была зафиксирована впервые.

Выходные данные:

Программа должна вывести на экран следующие результаты:

Средняя температура: [значение]

Количество дней с отрицательной температурой: [значение]

Максимальная температура: [значение], День: [значение]

Минимальная температура: [значение], День: [значение]

Решение.

```
temperatures = []
for i in range(30):
    while True:
        try:
            temp = int(input(f"Введите температуру за день {i+1}: "))
            temperatures.append(temp)
            break
        except ValueError:
            print("Некорректный ввод. Пожалуйста, введите целое число.")
```

```
# Анализ данных
total_temperature = 0
negative_days = 0
max_temp = temperatures[0]
max_temp_day = 1
min_temp = temperatures[0]
min_temp_day = 1

for i, temp in enumerate(temperatures):
    total_temperature += temp

    if temp < 0:
        negative_days += 1

    if temp > max_temp:
        max_temp = temp
        max_temp_day = i + 1

    if temp < min_temp:
        min_temp = temp
        min_temp_day = i + 1

average_temperature = total_temperature / 30

# Вывод результатов
print(f"Средняя температура: {average_temperature}")
print(f"Количество дней с отрицательной температурой: {negative_days}")
print(f"Максимальная температура: {max_temp}, День: {max_temp_day}")
print(f"Минимальная температура: {min_temp}, День: {min_temp_day}")
```

Описание кода.

Первый шаг – получение данных о температуре от пользователя. Для этого создается пустой список под названием temperatures, в который будут добавляться значения температуры для каждого дня. Затем запускается цикл, который повторяется 30 раз – по разу для каждого дня месяца. Внутри этого цикла используется еще один цикл, while True, который гарантирует, что пользователь введет корректные данные, а именно – целое число.

Для обработки возможных ошибок при вводе используется конструкция try-except. В блоке try программа пытается преобразовать введенную пользователем строку в целое число с помощью функции int(). Если преобразование проходит успешно, то полученное число, представляющее температуру за определенный день, добавляется в список temperatures с помощью метода append(). После добавления температуры в список, внутренний цикл while True прерывается оператором break, и программа переходит к следующему дню. Если же пользователь вводит что-то, что не может быть преобразовано в целое число, например, текст или дробное число, возникает ошибка ValueError.

В этом случае, блок except перехватывает эту ошибку, выводит информационное сообщение о том, что нужно ввести целое число, и программа возвращается к началу цикла while True, чтобы пользователь мог

попробовать ввести данные снова. Таким образом, обеспечивается надежный ввод данных.

После того, как все данные о температуре успешно введены и сохранены в списке temperatures, начинается этап анализа данных. Сначала, инициализируются несколько переменных: total_temperature для подсчета общей суммы температур (она начинается с 0), negative_days для подсчета количества дней с отрицательной температурой (тоже начинается с 0), а также max_temp и min_temp для хранения максимальной и минимальной температур соответственно.

Важно отметить, что переменные max_temp и min_temp изначально устанавливаются равными первому элементу списка temperatures, и предполагается, что в списке есть хотя бы один элемент. Так же хранятся переменные max_temp_day, min_temp_day -- день, в котором зафиксирована максимальная и минимальная температуры соответственно. При этом max_temp_day и min_temp_day инициализируются значением 1, поскольку отсчет дней начинается с 1.

Затем запускается цикл for, который перебирает все элементы списка temperatures. Функция enumerate позволяет получить индекс i текущего элемента, который используется для определения номера дня. Для каждой температуры в списке выполняется несколько проверок. Сначала, текущая температура добавляется к общей сумме температур total_temperature. Затем, проверяется, является ли текущая температура отрицательной.

Если да, то увеличивается счетчик отрицательных дней negative_days. Далее, проверяется, больше ли текущая температура, чем текущее значение max_temp. Если это так, то max_temp обновляется текущей температурой, а также номер дня, в который она зафиксирована, сохраняется в переменной max_temp_day. Это происходит только в том случае, если найдена новая максимальная температура. Аналогично, проверяется, меньше ли текущая температура, чем текущее значение min_temp. Если это так, то min_temp обновляется, и только если найдена температура меньше текущей минимальной, в min_temp_day сохраняется номер дня, когда была зафиксирована минимальная температура.

После завершения цикла анализа, вычисляется средняя температура, путем деления общей суммы температур total_temperature на количество дней (30). Этот результат сохраняется в переменной average_temperature.

Наконец, программа выводит результаты анализа на экран. Используя f-строки, программа выводит значения средней температуры, количества дней с отрицательной температурой, максимальной температуры и дня, когда она была зафиксирована, а также минимальной температуры и дня, когда она была зафиксирована.

F-строки позволяют легко встраивать значения переменных в текст вывода, делая его более читаемым и понятным."

Литература для подготовки

1. Информатика: 11-й класс: учебник. Босова Л.Л., Босова А.Ю., АО «Издательство «Просвещение».
2. Информатика: 11-й класс: учебник. Гейн А.Г. АО «Издательство «Просвещение»

Информационные ресурсы:

1 Единая коллекция цифровых образовательных ресурсов [Электронный ресурс]. – Режим доступа http://school-collection.edu.ru
2 Сайт журнала информатика. [Электронный ресурс]. – Режим доступа Журнал «Информатика и образование» — Издательство "Образование и Информатика"