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 **8** 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 основ информатики и закономерностей работы за компьютером.

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 работы с позиционными системами счисления;
2. владеть навыками моделирования;
3.владеть навыками работы в табличном редакторе;
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. Кодирование информации

Кодирование — это представление информации в форме, удобной для её хранения, передачи и обработки. Правило преобразования информации к такому представлению называется **кодом**.

Несложно показать, что эта битовая цепочка декодируется однозначно. Действительно, первая буква — М (код 01), потому что ни одно другое кодовое слово не начинается с 01. Аналогично определяем, что вторая буква — А. Действительно, за 01 следует 00 (код буквы А) и никакое другое кодовое слово не начинается с 00. Это же свойство, которое называется условием Фано, выполняется не только для кодовых слов 01 и 00, но и кодовых слов всех других букв.

Условие Фано. Никакое кодовое слово не совпадает с началом другого кодового слова.

Задание построения неравномерного кода удобнее решать с помощью дерева: условие Фано выполняется тогда, когда все выбранные кодовые слова заканчиваются в листьях дерева. При этом, те кодовые слова, которые встречаются чаще должны иметь более короткую длину.

Равномерные коды

Необходимо разбить цепочку на отдельные кодовые слова равномерной длины. В нашем примере 6 символов, поэтому можно использовать 3-битный код (который позволяет закодировать $8 = 2^3$ различных символов).

Пример задания:
Закодируем фразу из примера 1, используя код:

М	**А**	**Ы**	**Л**	**У**	**пробел**
000	001	010	011	100	101

Получаем закодированное сообщение

МАМА МЫЛА ЛАМУ → 000001000001101000010011001101011001000100

Длина этого сообщения — 42 бита, его легко разбить на отдельные кодовые слова и раскодировать («_» обозначает пробел):

000 001 000 001 101 000 010 011 001 101 011 001 000 100

М А М А _ М Ы Л А _ Л А М У

Видим, что равномерные коды неэкономичны (закодированное сообщение в примере 2 почти в два раза длиннее, чем в примере 1), но зато декодируются однозначно!

Ответ: 42 бита

Неравномерные коды

Для того, чтобы сократить длину сообщения, можно попробовать применить неравномерный код, то есть код, в котором кодовые слова, соответствующие разным символам исходного алфавита, могут иметь разную длину.

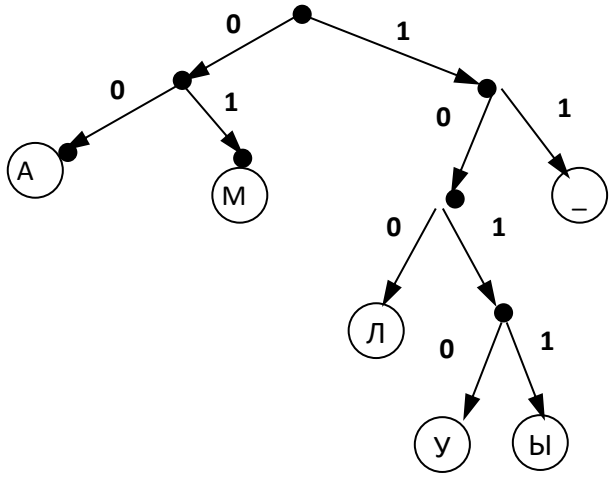
Пример задания:
Используем для кодирования фразы из примера 1 следующий код:

М	**А**	**Ы**	**Л**	**У**	**пробел**
01	00	1011	100	1010	11

Получаем

01	00	01	00	11	01	1011	100	00	11	100	00	01	11
М	А	М	А		М	Ы	Л	А		Л	А	М	У

Здесь 34 бита. Это, конечно, не 22, но и не 42.



Ответ: 34 бита

Тема 2. Двоичная арифметика

Правила выполнения арифметических действий над двоичными числами такие же, как и в десятичной системе, и задаются таблицами двоичного сложения, вычитания и умножения.

Арифметические действия над двоичными числами

Таблица двоичного сложения	Таблица двоичного вычитания	Таблица двоичного умножения
0+0=0	0-0=0	0×0=0
0+1=1	1-0=1	0×1=0
1+0=1	1-1=0	1×0=0
1+1=10	10-1=1	1×1=1

При сложении двоичных чисел производится сложение цифр слагаемых в каждом разряде и единиц переноса из соседнего младшего разряда, если они имеются. При этом необходимо учитывать, что в двоичной системе переполнение разряда наступает при количестве единиц, больше либо равным двум. В случае переполнения нужно вычесть из текущего разряда число, равное основанию системы (в данном случае – два), и добавить единицу переноса в следующий старший разряд.

Пример задания:

Выполнить сложение двоичных чисел (X, Y, Z):

а) X=1101, Y=101;

единицы переноса

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

Результат: 1101+101=10010.

Ответ: 10010

Пример задания*:*

X=1101, Y=101, Z=111;

$$\begin{array}{cccccc} & & {\scriptstyle 1} & & & \\ & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & \\ & 1 & 1 & 0 & 1 & \\ + & & & & & \\ & & 1 & 0 & 1 & \\ + & & & & & \\ & & 1 & 1 & 1 & \\ \hline 1 & 1 & 0 & 0 & 1 & \end{array}$$

Результат: 1101+101+111=11001.

Ответ: 11001

При вычитании двоичных чисел, аналогично вычитанию десятичных, может возникнуть необходимость займа единицы из предыдущего старшего разряда. Эта занимаемая единица переносится в текущий разряд как двойка (количество единиц, равное основанию).

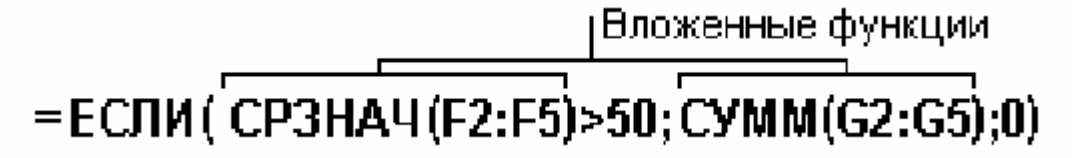
Тема 3. Работа с табличным процессором

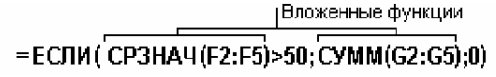
Функции — заранее определенные формулы, которые выполняют вычисления по заданным величинам, называемым аргументами, и в указанном порядке, определяемом синтаксисом.

Функции MS Excel позволяют выполнять как простые, так и сложные вычисления, связанные с решением определенных задач. Некоторые вычисления могут быть выполнены как с помощью формул, так и с помощью аналогичных им функций. Например: Формула =C7+D7+E7 складывает содержимое ячеек C7, D7 и E7.

Пример задания:

Функция =СУММ(C7:E7) — выполняет то же самое. Функции, используемые в программе MS Excel, имеют следующий синтаксис: =ФУНКЦИЯ (аргумент1,аргумент2,...) В некоторых случаях может потребоваться использование функции как одного из аргументов другой функции. Например, на Рис. 4.3 показано, как функции СРЗНАЧ и СУММ вложены в функцию ЕСЛИ.



Ответ: 

Пример задания:
В ячейках столбца С требуется найти сумму ячеек столбцов А и В. Для этого введите в ячейку С1 формулу, как показано на рисунке 2 и нажмите клавишу Enter.

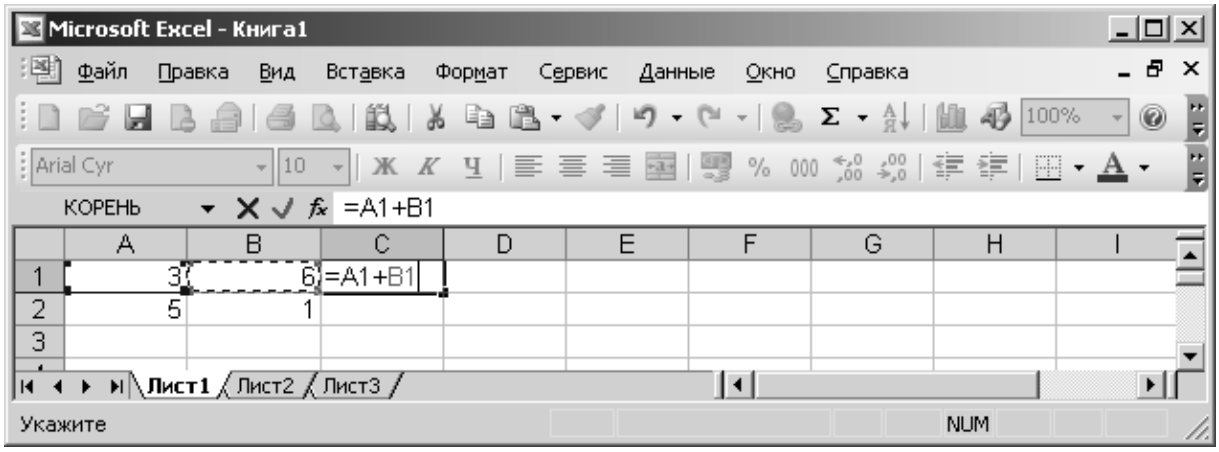


Рисунок 1 – Ввод формулы с относительными ссылками

Введенная формула показывает, что необходимо **относительно** текущей ячейки (ячейки С1) сложить значения, содержащиеся в двух соседних слева ячейках (ячейки А1 и В1).

Для того, чтобы в ячейке С2 получить сумму ячеек А2 и В2 достаточно просто скопировать формулу из ячейки С1 в ячейку С2. Копирование можно выполнить двумя способами: через буфер обмена или при помощи маркера заполнения (в этом случае следует навести указатель мыши на правый нижний угол ячейки С1, так чтобы указатель мыши изменился на +). Поскольку в ячейке С1 формула с относительными адресами, то в ячейку С2 скопируется формула =A2+B2 (рисунок 3)

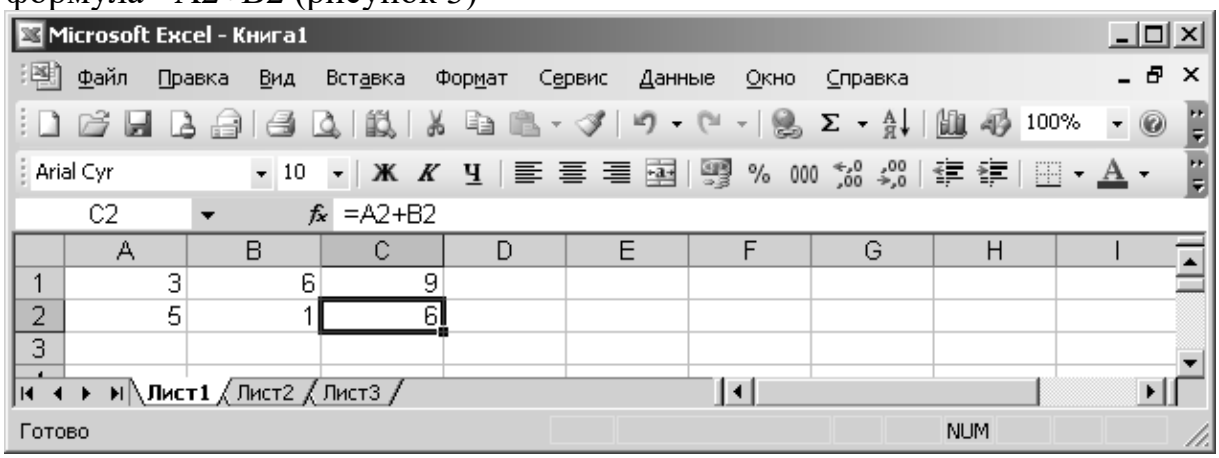


Рисунок 2 – Копирование относительной ссылки по столбцу вниз

На рисунке 3 показано, что относительные адреса при копировании корректируются. Для закрепления материала скопируйте формулу из ячейки C1 в ячейку D1 (рисунок 4).

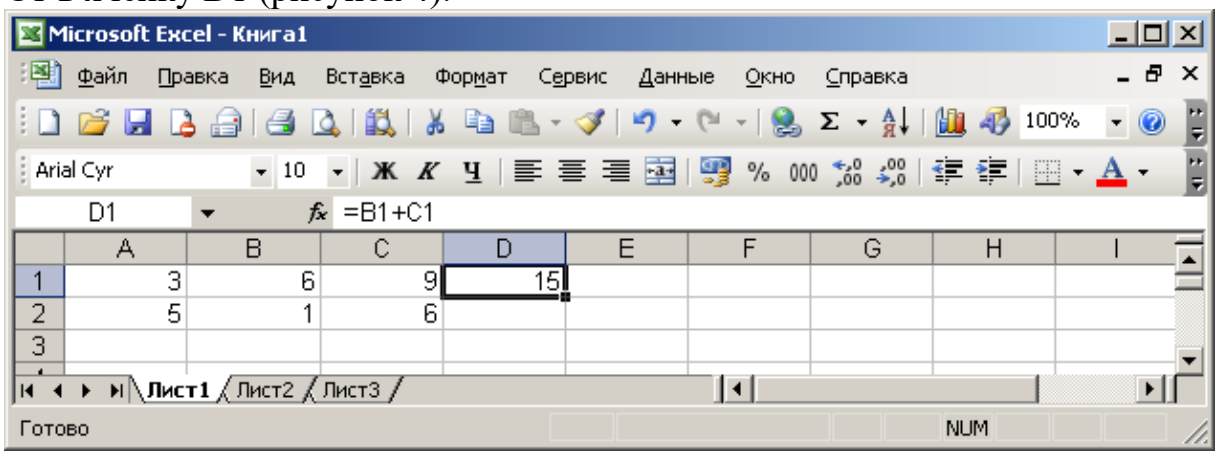


Рисунок 3 – Копирование относительной ссылки по строке влево

Как видно из рисунка 4 в ячейке D1 при копировании получена формула =B1+C1, т.е. сумма двух соседних слева ячеек относительно активной ячейки D1.

Ответ: B1+C1

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: 8-й класс: учебник. Босова Л.Л., Босова А.Ю., АО «Издательство «Просвещение».
2. Информатика: 8-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"