Сойникова Екатерина Дмитриевна

МКОУ «Новоспасская средняя общеобразовательная школа»

Золотухинского района Курской области

учитель информатики

**Программирование на Python как альтернатива теоретическому решению заданий ОГЭ и ЕГЭ по информатике**

В современном обучении информатике программирование занимает важное место, особенно в контексте подготовки к экзаменам по информатике, таким как ОГЭ и ЕГЭ. Одним из ключевых инструментов, который может значительно упростить решение задач, является язык программирования Python.

Python обладает рядом преимуществ, которые делают его привлекательным для использования в образовательных целях. Во-первых, он прост в изучении и использовании, что позволяет даже начинающим программистам быстро освоить основные алгоритмические структуры и структуры данных. Во-вторых, Python широко используется в различных областях, таких как наука о данных, машинное обучение и веб-разработка, что делает его универсальным инструментом для будущих специалистов.

В рамках подготовки к ОГЭ и ЕГЭ по информатике Python может быть использован для более быстрого решения задач, что значительно сокращает время на выполнение заданий и позволяет сосредоточиться на более сложных задачах. Например, задачи на обработку массивов данных, сортировку и поиск могут быть легко реализованы с использованием Python, что упрощает процесс проверки и анализа результатов.

В данной практике будут представлены возможности использования Python для решения заданий ОГЭ и ЕГЭ по информатике, а также приведены примеры задач, которые могут быть решены с помощью данного языка программирования. Я также проанализирую преимущества и недостатки использования Python в контексте подготовки к экзаменам и предложу рекомендации по его эффективному применению в образовательном процессе.

Таким образом, данная педагогическая практика направлена на демонстрацию потенциала Python как альтернативного инструмента для решения задач ОГЭ и ЕГЭ по информатике, что может способствовать повышению эффективности подготовки и улучшению результатов обучающихся.

ОГЭ по информатике является одним из самых легких экзаменов, среди всех предметов, предложенных в 9 классе экзаменов по выбору. Поэтому с каждым годом все чаще всего выпускники 9 класса предпочитают сдавать информатику.

Задачи ОГЭ по информатике в основном подразумевают письменное решение, либо же решение с использованием прикладных программ, например текстовых редакторов или электронных таблиц. При этом умение программировать до 2024 года требовалось только в последнем задании под номером 15. Которое в свою очередь предоставляло школьнику выбор, а именно в 15.1 требовалось умение создавать и выполнять программы для заданного исполнителя (Робота), а в 15.2 необходимо умение создавать и выполнять программы на универсальном языке программирования. В 2025 году, согласно новой демоверсии ОГЭ по информатике эти два задания разделили, и теперь всего в экзамене 16 заданий, последним из которых является то самое задание на выполнение программы на универсальном языке программирования.

До 2024 года, чтобы получить оценку «отлично» необходимо было набрать 16-19 баллов, таким образом выпускники, желающие получить положительную оценку выполняли либо задание №15.1, где необходимо написать программу на достаточно легком для изучения Школьном алгоритмическом языке. Лишь малая часть обучающихся выбирало писать программу на универсальном языке программирования в номере 15.2. При этом стоит отметить, что из тех ребят, которые все-таки выбрали решить задание 15.2 большинство решает данный номер на языке программирования Pascal, которые до 2023 года изучался в 8-9 классе как единственный и основной (в учебниках Л.Л. Босовой в ред. От 2023 года предложено помимо Pascal изучать Python).

Причиной того, что школьники не решают задачу на выполнение программы на универсальном языке программирования, является недостаточное количество часов на изучение высокоуровневых языков программирования в курсе информатики. Из-за чего дети на уроках учатся выполнять задачи на программирование по данным в учебниках шаблонам, но при этом не понимают, как можно использовать программирование при решении различных задач информатики. Все это приводит к тому, что обучающиеся, которые выбрали сдавать в 11 классе уже ЕГЭ по информатике, сталкиваются с огромными трудностями в том, что им необходимо решить достаточно сложные задачи, которые «ручным» решением выполнить очень проблематично и долго.

Таким образом, в 9 классе при подготовке к ОГЭ по информатике уже стоит обращать внимание обучающихся на то, что некоторые задачи можно решать при помощи программирования.

Рассмотрим примеры заданий ОГЭ по информатике, которые возможно решить при помощи программирования.

Задание № 3 направлено на проверку умения определять истинность составного высказывания. Данное задание, наряду с заданием № 6, о котором будет сказано позже является одним из самых сложных заданий части 1 ОГЭ по информатике.

Все задания № 3 можно глобально разделить на подтипы.

Подтип 1.Поиск наибольшего значения x, для которого истинно/ложно логическое высказывание.

Пример 1 (Открытый банк заданий ФИПИ). Определите наибольшее натуральное число **x**, для которого логическое выражение **ложно**:

**НЕ** ((x < 8) **И** (x < 21)) **ИЛИ** (x нечётное).

Решение на языке Python

Через цикл с заданным числом повторений запустим перебор всех значений x, которые будут удовлетворять условию задачи, т.е. давать ложное значение для логического выражения:

**for** x **in** range(1,100):

**if** (**not**((x < 8) **and** (x < 21)) **or** (x % 2 == 1)) == **False**:

print(x)

При запуске программы необходимо выбрать наибольшее из получившихся значений, т.е. x = 6.

Пример 2 (Открытый банк заданий ФИПИ). Напишите наибольшее трёхзначное число **x**, для которого **истинно** высказывание:

**НЕ** (Первая цифра нечётная) **И** (x делится на 3).

Решение на языке Python

**for** x **in** range(100,1000):

**if** (**not**(x // 100) % 2 != 0) **and** (x % 3 == 0) == **True**:

print(x)

Подтип 2. Поиск наименьшего значения x для которого истинно/ложно логическое высказывание.

Пример 3 (Открытый банк заданий ФИПИ). Определите наименьшее натуральное двузначное число x, для которого ложно логическое выражение:

**НЕ** (x нечётное) **И НЕ** (x > 88).

Решение на языке Python

**for** x **in** range(10,100):

**if** ((**not**(x % 2 == 1) **and not**(x > 88))) == **False**:

print(x)

**break**

В данной задаче оператора **break** обеспечивает выход из цикла, так как нам достаточно вывода самого первого значения x.

Подтип 3. Вычисление количества x для которого истинно/ложно логическое высказывание.

Пример 4 (Открытый банк заданий ФИПИ). Определите количество натуральных двузначных чисел x, для которых истинно логическое выражение:

**НЕ**(x чётное)**И НЕ**(x кратно 5).

Решение на языке Python

Так как в данной задаче требуют посчитать количество натуральных двузначных чисел x, то нам необходимо ввести переменную, которая и будет считать все удовлетворяющие условию x:

cnt=0 #счетчик

**for** x **in** range(10,100):

**if** (**not**(x % 2 == 0) **and not**(x % 5 == 0)) == **True**:

cnt+=1

print(cnt) #выводим только количество чисел

Подтип 4. Поиск одного числа из четырех, которое удовлетворяет условию.

Пример 5 (Открытый банк заданий ФИПИ). Дано четыре числа: 638, 442, 357, 123. Для какого из приведённых чисел истинно высказывание:

**НЕ**(Первая цифра чётная) **И НЕ** (Сумма цифр чётная)

В ответе запишите это число.

**for** i **in** range(4):

a = int(input('Введите число '))

n3 = a % 10 #поочередно отсекаем цифру от числа

n2 = a % 100 // 10

n1 = a % 1000 // 100

**if** (**not**(n1 % 2 == 0) **and not**((n1 + n2 + n3) % 2 == 0)) == **True**:

print(a)

Одни из наиболее легких в ОГЭ по информатике является задание № 5, где проверяется умение анализировать простые алгоритмы для конкретного исполнителя с фиксированным набором команд. Но среди данных заданий можно и также увидеть усложненные прототипы.

Пример 6 (Демовариант 2024 года). У исполнителя Альфа две команды, которым присвоены номера:

1. **прибавь 1**
2. **умножь на *b***

(*b* – неизвестное натуральное число; *b* ≥ 2).

Первая из них увеличивает число на 1, вторая умножает его на *b*. Алгоритм для исполнителя Альфа – это последовательность номеров команд. Найдите значение числа *b*, при котором из числа 6 по алгоритму 11211 будет получено число 82.

Решение на языке Python

**for** b **in** range (2,100):

x = 6 + 1 + 1#выполнение команды 1 над исходным числом

**if** (x \* b + 1 + 1) == 82: #выполнение остальных команд

print(b)

Перейдем к самому сложному заданию части 1 ОГЭ по информатике – заданию № 6.

Сложность задания 6 ОГЭ по информатике заключается в необходимости глубокого понимания основ программирования и алгоритмов, а также умения анализировать и применять их на практике. Данные задания также можно разделить на подтипы: без параметра и с параметром A. Если без параметра A – это обычная задача на анализ программы, то с параметром задача гораздо становится сложнее.

Пример 7 (Открытый банк заданий ФИПИ). Ниже приведена программа, записанная на пяти языках программирования.

|  |  |
| --- | --- |
| **Алгоритмический язык** | **Паскаль** |
| алг  нач  цел s, t, A  ввод s  ввод t  ввод A  если s > A или t > 12  то вывод "YES"  иначе вывод "NO"  все  кон | var s, t, A: integer;  begin   readln(s);   readln(t);   readln(A);   if (s > A) or (t > 12)    then writeln("YES")    else writeln("NO")  end. |
| **Бейсик** | **Python** |
| DIM s, t, A AS INTEGER  INPUT s  INPUT t  INPUT A  IF s > A OR t > 12 THEN    PRINT "YES"  ELSE    PRINT "NO"  ENDIF | s = int(input())  t = int(input())  A = int(input())  if (s > A) or (t > 12):      print("YES")  else:      print("NO") |
| **C++** | |
| #include <iostream>  using namespace std;    int main(){     int s, t, A;     cin >> s;     cin >> t;     cin >> A;     if (s > A || t > 12)       cout << "YES" << endl;     else       cout << "NO" << endl;     return 0;   } | |

Было проведено 9 запусков программы, при которых в качестве значений переменных *s* и *t* вводились следующие пары чисел:

(13, 2); (11, 2); (–12, 12); (2, –2); (–10, –10); (6, –5); (2, 8); (9, 10); (1, 13).

Укажите наименьшее целое значение параметра *А*, при котором для указанных входных данных программа напечатает «NO» восемь раз.

Решение на Python будет выглядеть следующим образом. Для начала необходимо задать функцию с тремя переменными s, t и A, после чего описать само условие, данное в задаче. Оператор **return** возвращает данные в функции после выполнения работы самой функции. После чего прописываем запуски программы, вводим переменную «счетчика», которая будет считать количество напечатанных «NO»

**def** f(s, t, A): #задание функции с тремя переменными

**if** (s > A) **or** (t > 12): #описание условия

**return** 0

**else**:

**return** 1

a = ((13, 2), (11, 2), (-12, 12), (2, -2), (-10, -10), (6, -5), (2, 8), (9, 10), (1, 13)).

**for** A **in** range(100):

cnt = 0

**for** i **in** range(9):

cnt += f(a[i][0], a[i][1], A) #результат работы цикла

**if** cnt == 8:

print(A)

**break**

Для решения данного типа задания при помощи программирования школьник должен обладать достаточно большими знаниями, понимать то, как задавать функцию и то, как она работает. Но, к сожалению, на изучение функций в курсе программирования в 9 классе уделяется очень мало времени, а именно, согласно ФРП по информатике, на данную тему выделен лишь 1 час.

Поэтому школьникам, сдающим ОГЭ по информатике важно обращать дополнительно особое внимание на программирование процедур и функций, так как при решении заданий ЕГЭ с помощью программирования, часто приходится прибегать к использованию функций в программном коде.

Стоит таже рассказать о том, что существуют и встроенные функции.

Например, в учебнике для 9 класса Л.Л. Босовой при изучении вспомогательных алгоритмов на Python предлагается выполнить задание на написание функции, которая вычисляет количество цифр в двоичной записи вводимого десятичного числа. При этом, главное, что необходимо выполнить для решения данной задачи – это написать программу, которая будет переводить число из десятичной системы счисления в двоичную:

**def** two(a):

b = '' #создаем пустую строку

**while** a > 0: #пока число a > 0

b = **str**(a % 2) + b #заполняем пустую строку остатками от деления a на 2

a = a // 2 #забираем целую часть

b = int(b) #перевод строки в число

print('Двоичная система счисления: ',b)

a = int(input('Введите число ')) #вводим число, которое хотим перевести

a2 = two(a)#применяем к полученному числу написанную функцию

Затем обучающимся можно показать, что написанную ими функцию можно заменить на встроенную функцию **bin**, которая так же переводит число из десятичной системы счисления в двоичную. Тогда программа будет иметь вид:

a = int(input('Введите число '))

print('Двоичная система счисления: ',bin(a)[2:])

Также стоит сказать, что в Python существуют специальные функции, позволяющие переводить десятичные числа в восьмеричную и шестнадцатеричную системы счисления:

print(oct()) – для восьмеричной

print(hex)) – для шестнадцатеричной

Таким образом, можно перейти к решению номера 10 ОГЭ по информатике при помощи программирования. Данное задание направлено на проверку умения записывать числа в различных системах счисления.

Важно подчеркнуть, что переводить числа можно не только из десятичной в какую-либо систему счисления, но и из какой-либо системы счисления в десятичную.

Для того, чтобы перевести число в десятичную систему счисления в любую другую достаточно использовать функцию **int** при этом нужно строку перевести в число. Например, чтобы перевести число 100010 из двоичной системы счисления необходимо записать следующее:

print(int("100010", 2))

Пример 8 (Открытый банк заданий ФИПИ). Переведите число 105 из десятичной системы счисления в двоичную систему счисления.

На языке Python решение будет выглядеть следующим образом:

print(bin(105))

Запустив программу, мы получим ответ и два технических символа впереди:

0b1101001

Их необходимо убрать в получившейся строке. Тогда «заберем» эти два символа, сделав «срез строки»:

print(bin(105)[2:])

После изучения теоретического материала можно предложить выполнить ряд следующих заданий.

Пример 9 (Открытый банк заданий ФИПИ). Вычислите значение арифметического выражения:

110110112 + 11108 – 11116

В ответе запишите десятичное число, основание системы счисления указывать не нужно.

Однострочное решение на языке Python:

print(int('11011011',2) + int('1110',8) - int('111',16))

Ответ: 530.

Пример 10 (Открытый банк заданий ФИПИ). Переведите число 297 из десятичной системы счисления в двоичную систему счисления. Сколько единиц содержит полученное число?

В ответе укажите одно число – количество единиц.

Для решения данного типа задач с помощью программирования необязательно сначала переводить число в двоичную систему счисления, после чего «вручную» считать количество единиц. Так как мы работаем со строкой, то для того, чтобы подсчитать количество того или иного символа в строке, используем функцию **count**. Для нашего примера это будет выглядеть следующим образом:

print(bin(297)[2:].count("1"))

Ответ: 193.

Пример 11 (Демонстрационный вариант контрольных измерительных материалов основного государственного экзамена 2024 года по информатике). Среди приведенных ниже трёх чисел, записанных в различных системах счисления, найдите максимальное и запишите его в ответе десятичной системе счисления. В ответе запишите только число, основание системы счисления указывать не нужно.

2316, 328, 111102

В данной задаче для поиска максимального числа воспользуемся функцией **max**:

print(max(int('23',16),int('32',8),int('11110',2)))

Ответ: 35.

При подготовке к ОГЭ по информатике стоит так же показать, как с помощью программирования решать более сложные задачи на системы счисления, приближенные к заданиям ЕГЭ по информатике.

Пример 12 (Открытый банк заданий ФИПИ). ​Вычислите значение суммы: 128 + 1210 + 1216. Результат представьте в виде двоичного числа.

При решении данной задачи необходимо привести все числа одну систему счисления, например, в десятичную:

print(int('12',8) + 12 + int('12',16))

После чего дополнить код, и перевести данную запись в двоичную систему счисления:

print(bin(int('12',8) + 12 + int('12',16))[2:])

​Ответ: 101000

Пример 13 (Открытый банк заданий ФИПИ).​ Для скольких натуральных чисел X выполняется неравенство:  ?

Переведем числа и в десятичную систему счисления:

int('b6',16)

int('275',8)

Так как нам необходимо найти количество натуральных *X*, которые входят в заданный промежуток, то введем переменную, обозначающую «счетчик», после чего при помощи цикла с заданным числом повторений переберем все *X*, которые будут удовлетворять заданному условию:

cnt = 0

**for** x **in** range(1000):

**if** x > int('b6',16) **and** x < int('275',8):

cnt +=1

print(cnt)

​После запуска программы в ответе получим 6 чисел.

Ответ: 6.

Пример 14 (Открытый банк заданий ФИПИ).​ Укажите через запятую (без пробела) в порядке возрастания все числа, не превосходящие 30, в запись которых в двоичной системе счисления оканчивается на «101». Числа в ответе указывайте в десятичной системе счисления.

С помощью цикла с заданным числом повторений переберем все числа, которые меньше 30. Затем переведем их в двоичную систему счисления. Создадим условие «если три последние цифры двоичной записи – «101», то результат переведем в десятичную систему счисления и выведем все подходящие числа.

**for** n **in** range(30):

n2 = bin(n)[2:]

**if** n2[-3:] == '101':

R = int(n2,2)

print(n)

Ответ: 5, 13, 21, 29.

Таким образом, научившись решать задания ОГЭ по информатике с помощью программирования на Python, школьники смогут понять, что их ждет при подготовке к ЕГЭ.

Рассмотрим примеры прототипов заданий ЕГЭ по информатике, которые возможно решать с помощью программирования на Python, помимо заданий, которые требуют умения писать программы.

Первое задание, которое целесообразно решать при помощи программирования – это задание № 2.

Пример 14 (Открытый банк заданий ФИПИ).​ Миша заполнял таблицу истинности функции

(¬*x* /\ ¬*y*)\/ (*y* ≡ *z*) \/ *w*,

но успел заполнить лишь фрагмент из трёх различных её строк, даже не указав, какому столбцу таблицы соответствует каждая из переменных *w*,*x*,*y*,*z*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  | **(¬*x* /\ ¬*y*)\/ (*y*≡*z*) \/ *w*** |
|  |  | 1 |  | **0** |
| 1 | 0 |  | 1 | **0** |
| 0 | 0 | 1 | 1 | **0** |

Определите, какому столбцу таблицы соответствует каждая из переменных *w*,*x*,*y*,*z*.

В ответе напишите буквы *w*,*x*,*y*,*z* в том порядке, в котором идут соответствующие им столбцы (сначала буква, соответствующая первому столбцу; затем буква, соответствующая второму столбцу, и т.д.). Буквы в ответе пишите подряд, никаких разделителей между буквами ставить не нужно.

*Пример.* Функция задана выражением ¬*x* \/ *y*, зависящим от двух переменных, а фрагмент таблицы имеет следующий вид.

|  |  |  |
| --- | --- | --- |
|  |  | **¬*x* \/ *y*** |
| 0 | 1 | **0** |

В этом случае первому столбцу соответствует переменная *y*, а второму столбцу –– переменная *x*. В ответе следует написать: *yx*.

Решение на Python достаточно простое, и подходит для всех прототипов данного задания. В разных источниках существуют разные варианты программы для номера 2 ЕГЭ по информатике. Я обучаю школьников писать достаточно понятный и простой. Первое, что необходимо, это задать функцию с четырьмя переменными x, y, z и w, после чего вернуть само логическое высказывание, данное в условии задачи. Следующий шаг – это перебор самих переменных x, y, z и w, при которых данное логическое высказывание возвращает ложь или истину (в зависимости от данных таблицы). И вывод самого фрагмента таблицы истинности.

**def** f(x, y, z, w):

**return** (((**not**(x)) **and** (**not**(y))) **or** (y == z) **or** w)

print('x y z w F')

**for** x **in** range(2):

**for** y **in** range(2):

**for** z **in** range(2):

**for** w **in** range(2):

**if** f(x, y, z, w) == **False**:

print(x, y, z, w, **int**(f(x, y, z, w)))

Результат работы программы (см. рис. 1).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Результат работы программы

Сопоставив результат работы программы с таблицей, данной в условии задачи, становится понятно, что все нули могут быть только во втором столбце таблицы, где стоит уже два нуля, следовательно, переменная w будет стоять во втором столбце таблицы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **w** |  |  | **(¬*x* /\ ¬*y*)\/ (*y*≡*z*) \/ *w*** |
|  | **0** | 1 |  | **0** |
| 1 | 0 |  | 1 | **0** |
| 0 | 0 | 1 | 1 | **0** |

Посмотрев на вторую и третью строку таблицы, видно, что в них обеих по две цифры 1, следовательно, смотрим на результат программы, и видим, что в первой строке одна цифра 1, а в двух остальных их по две. В первой строке единице равна только переменная y, заполним таблицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **w** | **y** |  | **(¬*x* /\ ¬*y*)\/ (*y*≡*z*) \/ *w*** |
| **0** | **0** | 1 | **0** | **0** |
| 1 | 0 |  | 1 | **0** |
| 0 | 0 | 1 | 1 | **0** |

Теперь посмотрим на столбец z в результате работы программы, видим, что там два нуля и одна единица, следовательно, первый столбец таблицы условия – это переменная z, тогда последний столбец – это x.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **z** | **w** | **y** | **x** | **(¬*x* /\ ¬*y*)\/ (*y*≡*z*) \/ *w*** |
| **0** | **0** | 1 | **0** | **0** |
| 1 | 0 | **0** | 1 | **0** |
| 0 | 0 | 1 | 1 | **0** |

Ответ: zwyx.

Задание № 4 ЕГЭ по информатике одно из самых легких, как отмечают школьники. При этом оно является одним из самых коварных, как правило именно в нем делают ошибку даже претенденты на высокий балл. Скорее всего это связно с тем, что обучающиеся могут невнимательно прочитать условие задачи или неверно интерпретировать вопрос в условии.

Чаще всего данное задание решают «ручным» решением методом построения дерева, на котором размещаются буквы. Это достаточно простой и быстрый способ. Однако, данную задачу можно решать и с помощью программного кода.

Рассмотрим решение некоторых прототипов задания 4 при помощи программирования на примере следующей задачи:

Пример 15 (Открытый банк заданий ФИПИ).​ По каналу связи передаются сообщения, содержащие только четыре буквы: А, Б, В, Г, Д, Е; для передачи используется двоичный код, удовлетворяющий условию Фано. Для букв А, Б, В, Г, Д используются такие кодовые слова: А – 111; Б – 011; В – 10; Г – 001; Д – 000. Укажите кратчайшее кодовое слово для буквы Е, при котором код будет допускать однозначное декодирование. Если таких кодов несколько, укажите код с наибольшим числовым значением.

Решить программированием данную задачу можно методом перебора префиксов. Таким образом, запишем все известные нам кодовые слова:

codes = ['111', '011', '10', '001', '000']

Так как длина префиксов от 1 до 3 запишем

for i in range(1, 4):

а все префиксы являются по сути двоичными числами, где функция **zfill** дополняет строку нулями слева до указанной минимальной длины

**for** x **in** range(2\*\*i):

pfx = bin(x)[2:].zfill(i)

Таким образом, скажем, что если префикс не является началом другого, уже заданного префикса и заданные коды не являются началом префикса:

**if all**(c[:i] != pfx **and** pfx[:len(c)] != c **for** c **in** codes):

print(pfx)

**break**

Пример 16 (Сборник С. С. Крылова ЕГЭ-2023). По каналу связи передаются шифрованные сообщения, содержащие только 9 букв: А, Б, В, Г, Д, Е, Ж, З, И; для передачи используется неравномерный двоичный код. Для девяти букв используются кодовые слова.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Буква | Кодовое слово |  | Буква | Кодовое слово |
| А | 000 |  | Е | 010 |
| Б | 001 |  | Ж | 011 |
| В | 1110 |  | З | 11001 |
| Г | 11111 |  | И |  |
| Д | 1011 |  |  |  |

Укажите кратчайшее кодовое слово для буквы И, при котором код будет удовлетворять условию Фано. Если таких кодов несколько, укажите код с наименьшим числовым значением.

*Примечание.* Условие Фано означает, что никакое кодовое слово не является началом другого кодового слова. Это обеспечивает возможность однозначной расшифровки закодированных сообщений.

Решение на Python:

​codes = ['000', '001', '1110', '11111', '11000', '010', '011', '11001']

**for** i **in** range(1, 6):

**for** x **in** range(2 \*\* i):

prfx = **bin**(x)[2:].zfill(i)

**if all**(c[:i] != prfx **and** prfx[:len(с)] != c **for** c **in** codes):

print(prfx)

**break**

В результате работы программы мы видим четыре варианта кодовых слов для буквы И, в условии требуется выбрать кратчайший код. Таким образом буква И будет иметь код 10 (см рис. 2).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2 – Результат работы программы

Задание № 5 достаточно сложное для школьников. Согласно аналитическому отчёту о результатах участников ЕГЭ 2023 года по информатике от ФИПИ, примерно 35% выпускников, сдающих ЕГЭ по информатике, выполнили задание 5. Это говорит о том, что только третья часть школьников знает методику выполнения данного типа задач.

В целом прототипы задания № 5 можно разделить на две большие группы – в первой группе задачи, в которых строиться двоичная запись числа, а во второй – троичная запись числа. Принципиальная разница состоит в том, что для того, чтобы перевести число в двоичную систему счисления, необходимо применить встроенную функцию **bin.** Но для перевода в троичную систему счисления, функцию необходимо уже прописать самостоятельно, в чем и есть сложность для обучающихся. Тут как раз и стоит вспомнить задания на системы счисления, которые решались в 9 классе, а именно, функцию для перевода из десятичной системы счисления в любую другую.

Рассмотрим примеры.

Пример 17 (Открытый банк заданий ФИПИ).​ На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.

1. Строится двоичная запись числа *N*.

2. К этой записи дописываются справа ещё два разряда по следующему правилу:

а) складываются все цифры двоичной записи числа *N*, и остаток от деления суммы на 2 дописывается в конец числа (справа). Например, запись 11100 преобразуется в запись 111001;

б) над этой записью производятся те же действия – справа дописывается остаток от деления суммы её цифр на 2.

Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа *N*) является двоичной записью искомого числа *R*.

Укажите минимальное число *R*, которое превышает число 396 и может являться результатом работы данного алгоритма.

В ответе это число запишите в десятичной системе счисления.

**for** n **in** range(1,1000): #перебираем все n от 1 до 1000  
 n2 = **bin**(n)[2:] #стоим двоичную запись числа n  
 n2 += **str**(n2.count('1') % 2) # складываются все цифры двоичной записи числа N, и остаток от деления суммы на 2 дописывается в конец числа (справа).  
 n2 += **str**(n2.count('1') % 2)  
 R = **int**(n2,2) #результат переводим в десятичную систему счисления  
 **if** R > 396: #проверяем условие для R  
 print(R)  
 **break**

Ответ: 402

Пример 18 (Открытый банк заданий ФИПИ).​ На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.

1. Строится двоичная запись числа *N*.

2. К этой записи дописываются справа и слева ещё по одному или два разряда по следующему правилу: если *N* чётное, то в конец числа (справа) дописывается нуль, а в начало числа (слева) дописывается единица; если *N* нечётное, то в конец числа (справа) и в начало числа (слева) дописываются по две единицы.

*Пример*. Для числа 13 двоичная запись 1101 преобразуется в запись 11110111.

Полученная таким образом запись (в ней на два или четыре разряда больше, чем в записи исходного числа *N*) является двоичной записью искомого числа *R*.

Укажите **наименьшее** число *R*, превышающее 52, которое может являться результатом работы данного алгоритма.

В ответе это число запишите в десятичной системе счисления.

Решение на Python.

a = [] #задаем пустой массив, который будет заполнен результатами работы алгоритма (R)

**for** n **in** range(1,1000):  
 n2 = **bin**(n)[2:]  
 **if** n % 2 == 0: #проверка условия четности числа  
 n2 = '1' + n2 + '0'  
 **else**:  
 n2 = '11' + n2 + '11'  
 R = **int**(n2,2)  
 **if** R > 52:  
 a.append(R) #заполняем массив, если выполнено условие для R  
 print(**min**(a)) #выводим минимальный элемент массива

Ответ: 56

Задание пустого массива в данном типе задач необходимо из-за того, что если вывести просто значения R, то они не будут в порядке возрастания, поэтому из огромного числа элементов придется выбрать минимальный, что достаточно сложно сделать визуально. Поэтому все результаты заполняются в массив, после чего мы запрашиваем минимальный элемент этого массива.

Пример 19 (Открытый банк заданий ФИПИ).​На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.

1. Строится двоичная запись числа *N*.

2. К этой записи дописываются справа ещё несколько разрядов по следующему правилу:

а) если *N* чётное, то к нему справа приписываются два нуля, а слева единица;

б) если *N* нечётное, то к нему справа приписывается в двоичном виде сумма цифр его двоичной записи;

Полученная таким образом запись (в ней как минимум на один разряд больше, чем в записи исходного числа *N*) является двоичной записью искомого числа *R*.

*Например,*исходное число 410 = 1002 преобразуется в число 1100002= 4810, а исходное число 1310 = 11012 преобразуется в число 1101112= 5510.

Укажите такое **наименьшее** число *N*, для которого число *R* больше числа 190. В ответе запишите это число в десятичной системе счисления.

Решение на Python аналогично предыдущим примерам

**for** n **in** range(1,1000):  
 n2 = **bin**(n)[2:]  
 **if** n % 2 == 0:  
 n2 = '1' + n2 + '00'  
 **else**:  
 n2 = n2 + **str**(n2.count('1'))[2:] #сумма цифр в двоичной записи числа, т.е. кол-во единиц  
 R = **int**(n2,2)  
 **if** R > 190:  
 print(n)  
 **break**

Ответ: 16

Также решаются задачи, где необходимо проверить условие делимости на 3.

Пример 20 (Открытый банк заданий ФИПИ).​На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.

1. Строится двоичная запись числа *N*.

2. Далее эта запись обрабатывается по следующему правилу:

а) если число *N* делится на 3, то к этой записи дописываются три последние двоичные цифры;

б) если число *N* на 3 не делится, то остаток от деления умножается на 3, переводится в двоичную запись и дописывается в конец числа.

Полученная таким образом запись является двоичной записью искомого числа *R*.

3. Результат переводится в десятичную систему и выводится на экран.

*Например*, для исходного числа 12 = 11002 результатом является число 11001002 = 100, а для исходного числа 4 = 1002 это число 100112 = 19.

Укажите максимальное число *R*, не превышающее 162, которое может быть получено с помощью описанного алгоритма. В ответе запишите это число в десятичной системе счисления.

Решение на Python:

d = []  
**for** n **in** range (1,1000):  
 n2 = **bin**(n)[2:]  
 **if** n % 3 == 0:  
 n2 = n2 + n2[-3:] #отсекаем 3 последние цифры числа  
 **else**:  
 n2 = n2 + bin((n % 3) \* 3)[2:]  
 R = **int**(n2, 2)  
 **if** R < 162:  
 d.append(R)  
print(**max**(d))

Ответ: 151

Рассмотрим теперь решение самого сложного прототипа, где необходимо построить троичную запись числа. Для этого необходимо для начала написать функцию для перевода десятичного числа в троичную систему счисления.

Пример 21 (Открытый банк заданий ФИПИ).​На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.

1. Строится троичная запись числа *N*.

2. Далее эта запись обрабатывается по следующему правилу:

а) если число *N* делится на 3, то к этой записи дописываются две последние троичные цифры;

б) если число *N* на 3 не делится, то остаток от деления умножается на 5, переводится в троичную запись и дописывается в конец числа.

Полученная таким образом запись является троичной записью искомого числа *R*.

3. Результат переводится в десятичную систему и выводится на экран.

*Например*, для исходного числа 11 = 1023 результатом является число 1021013 = 307, а для исходного числа 6 = 203 это число 20203 = 60.

Укажите **минимальное** число *N*, после обработки которого с помощью этого алгоритма получается число *R*, не меньшее 86.

**def** tri(x):  
 s = ""  
 **while** x > 0:  
 s += **str**(x % 3)  
 x = x // 3  
 **return** s[::-1] #возвращаем строку в обратном порядке  
**for** n **in** range(1,1000):  
 n3 = tri(n)  
 **if** n % 3 == 0:  
 n3 += n3[-2:]  
 **else**:  
 n3 += tri((n % 3) \* 5)  
 R = **int**(n3, 3)  
 **if** R >= 86:  
 print(n)  
 **break**

Ответ: 5

Рассмотрим более подробно методику решения задания № 6 ЕГЭ по информатике. До 2023 года данное задание можно было решать на Школьном алгоритмическом языке в среде программирования КуМир, но с 2024 года данную задачу необходимо решать на высокоуровневых языках программирования. Таким образом, задача гораздо усложнилась. В рассматриваемых ниже примерах будет приведено в качестве сравнения «ручное» решение, которое очевидно не удобно и нецелесообразно, так в данном задании необходимо строить рисунок строго на клетчатой бумаге, которой у школьника нет в наличии на экзамене.

Пример 22 (Открытый банк заданий ФИПИ). Исполнитель Черепаха действует на плоскости с декартовой системой координат. В начальный момент Черепаха находится в начале координат, её голова направлена вдоль положительного направления оси ординат, хвост опущен.  
При опущенном хвосте Черепаха оставляет на поле след в виде линии. В каждый конкретный момент известно положение исполнителя и направление его движения. У исполнителя существует две команды: **Вперёд *n*** (где *n* – целое число), вызывающая передвижение Черепахи на *n* единиц в том направлении, куда указывает её голова; **Направо *m*** (где *m* – целое число), вызывающая изменение направления движения на *m* градусов по часовой стрелке.

Запись **Повтори *k* [Команда1 Команда2 … Команда*S*]** означает, что последовательность из *S* команд повторится *k* раз (где *k* – целое число).

Черепахе был дан для исполнения следующий алгоритм:

**Направо 315 Повтори 7 [Вперёд 16 Направо 45 Вперёд 8 Направо 135]**.

Определите, сколько точек с целочисленными координатами будут находиться внутри области, которая ограничена линией, заданной алгоритмом. Точки на линии учитывать не следует.

**1 Способ. Теоретическое решение**

1. в начале Черепаха поворачивается на 315°, так как изначально перед выполнением программы Черепаха стоит в начале координат и смотрит вверх («на север»).
2. тело цикла выполняется 7 раз, то есть, Черепаха три раза обойдёт фигуру и пройдет еще две стороны; важно, что **линия замкнётся**
3. теперь нужно нарисовать параллелограмм со сторонами 16 и 8 (команды **Вперёд 16** и **Вперёд 8**); это можно сделать, например, на листе в клеточку.
4. Для того, чтобы нарисовать сторону длинной 16, можно провести следующие расчёты:

* очевидно, что данная сторона проходит под углом 45°, т.е. она является диагональю квадрата
* пусть одна клетка имеет длину 1, тогда ее диагональ будет равна
* так как длина стороны равна 16, то разделим 16 на , и получим
* тогда проведем прямую под углом 45°, и отметим на ней 11 целых точек

1. теперь остается повернуть Черепаху на 45° вправо
2. построим прямую равную 8, повернем Черепаху на 135° вправо, и достроим параллелограмм (см. рис. 3).

Изображение выглядит как линия, График, диаграмма, Параллельный

Автоматически созданное описание

Рисунок 3 – Построение элементов рисунка

1. так как у нас получился параллелограмм, то достаточно посчитать только количество целых точек в одном из рядов – их 7, а таких рядов у нас 11, поэтом получим
2. Ответ: 77.

**Способ 2. Перебор с помощью программы**

1. для составления программы придётся построить систему условий, которая описывает область внутри контура
2. уравнение правой границы параллелограмма: , нас интересует область
3. верхняя граница: , нас интересует область ,

коэффициент

1. нижняя граница: , нас интересует область , коэффициент
2. уравнение левой границы параллелограмма, очевидно, что , а нас интересует область
3. организуем цикл, в котором перебираем значения от -12 до 1 и значения от 0 до 21; для каждой пары () проверяем все четыре условия; если они выполнены, увеличиваем счётчик **count** на 1:

**from** math **import** \*  
cnt = 0  
**for** x **in** range (-12,1):  
 **for** y **in** range (0,21):  
 **if** x < 0 **and** y > -x **and** y < -x + 8 **and** x > -8 \* sqrt(2):  
 cnt += 1  
print(cnt)

1. Ответ: 77.

**Способ 3. Решение c использованием модуля turtle в Python + перебор**

1. в Python есть стандартный модуль **turtle**, в котором реализована «черепашья графика»; его можно использовать для того, чтобы построить заданную фигуру.

**from** turtle **import** \*#подключение модуля Черепаха  
**from** math **import** \*tracer(2) #ускорение анимации пера  
m = 15 #масштаб  
left(90)#развернуть Черепаху "на север"

переменная **m** задает масштаб – длину единичного отрезка на плоскости в пикселях

1. Пропишем алгоритм, заданный в условии задачи

right(315) #вправо на 315  
pendown() #опустить хвост  
**for** i **in** range(7):  
 forward(16 \* m) #вперед на 16 (умножаем на масштаб)  
 right(45) #вправо на 45  
 forward(8 \* m) #вперед на 8 (умножаем на масштаб)  
 right(135) #вправо на 135  
penup() #поднять хвост

1. аналогично способу 2 организуем цикл, в котором перебираем значения от -12 до 1 и значения от -1 до 21; для каждой пары () проверяем все четыре условия;

cnt = 0 #количество попавших точек   
**for** x **in** range (-20,0):  
 **for** y **in** range (0,20):

goto (x \* m, y \* m) #переместить Черепаху в координатную плоскость

**if** x < 0 **and** y > -x **and** y < -x + 8 **and** x > -8 \* sqrt(2):

1. зададим цвет и толщину точки, если она попала внутрь параллелограмма, и увеличим счётчик **count** на 1:

dot (3, 'red')# (толщина, цвет – красный) точка находится внутри параллелограмма

cnt += 1

1. иной цвет зададим для точки, если она не попала внутрь параллелограмма, например, черный

else:

dot (3, 'black')# черная точка не находится внутри параллелограмма

1. выведем счет точек по каждой из колонок, зададим цвет

write(count)

color('red') #количество красных точек по колонкам

mainloop()#создание графического интерфейса

1. результат работы программы (см. рис. 4):

Изображение выглядит как снимок экрана, текст, линия, График

Автоматически созданное описание

Рисунок 4 – Результат работы программы

1. Ответ: 77.

Полный листинг программы:

**from** turtle **impor**t \*  
**from** math **import** \*  
tracer(2)  
hideturtle()  
m = 15  
left(90)  
right(315)  
pendown()  
**for** i **in** range(7):  
 forward(16 \* m)  
 right(45)  
 forward(8 \* m)  
 right(135)  
penup()  
cnt = 0  
k = -tan(pi / 4)  
**for** x **in** range(-12,1):  
 **for** y **in** range(-1,20):  
 goto(x \* m, y \* m)  
 **if** (y > k \* x + 0.5 **and** y < k \* x + 8 **and** x < 0 **and** x > -8\*sqrt(2)):  
 dot(3, 'red')  
 cnt += 1  
 **else**:  
 dot(3, 'black')  
 write(count)  
 color('red')  
mainloop()

**Способ 4. Решение c использованием модуля turtle в Python**

Выполняем действия аналогично с предыдущему методу:

**from** turtle **import** \*  
m = 15  
tracer(0)  
left(90)  
right(315)  
pendown()  
**for** i **in** range(7):  
 forward(16 \* m)  
 right(45)  
 forward(8 \* m)  
 right(135)  
penup()  
**for** x **in** range (-100,100):  
 **for** y **in** range(-100,100):  
 goto(x\*m, y\*m)  
 dot(2)  
done()

Получим изображение фигуры. подсчёт точек с целочисленными координатами, оказавшихся внутри контура, проводится визуально (см. рис. 5).

**Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание**

Рисунок 5 – Результат работы программы

Еще одним достаточно сложным для обучающихся является задание № 8, его в 2023 году выполнило менее 30% от всех сдающих экзамен выпускников. Сложность этого задания заключается в слабых знаниях комбинаторики.

Для начала необходимо отметить, что для каждого типа задачи будет написан отдельный шаблон программного кода. Но при этом при решении с помощью программы на языке Python удобно использовать функции из модуля **itertools**:

**combinations** *– комбинации*, например,

**from** itertools **impor**t combinations

cmb = list(combinations('ABC', 2))

print( cmb )

// Результат: [('A', 'B'), ('A', 'C'), ('B', 'C')]

**permutations** –*перестановки*, например,

**from** itertools **import** permutations

cmb = list(permutations('ABC'))

print(cmb)

// Результат: [('A', 'B', 'C'), ('A', 'C', 'B'),

// ('B', 'A', 'C'), ('B', 'C', 'A'), ('C', 'A', 'B'),

// ('C', 'B', 'A')]

**product** – *декартово произведение* (все возможные слова заданной длины, составленные из данного алфавита), например:

**from** itertools **import** product

cmb = list(product('ABC',repeat=2))

print( cmb )

// Результат: [('A', 'A'), ('A', 'B'), ('A', 'C'), ('B', 'A'),

// ('B', 'B'), ('B', 'C'), ('C', 'A'), ('C', 'B'), ('C', 'C')]

Как видно из этих примеров, результат работы этих трёх функций – массив кортежей. В нём удобно работать с отдельными символами, но неудобно искать сочетания букв. Если нужно работать с сочетаниями букв, нужно «склеить» символы каждого кортежа в строки с помощью метода .join:

***Рассмотрим первый тип задачи на системы счисления:***

*Определите количество n-значных чисел, записанных в n-ричной системе счисления, в записи которых.......*

Таким образом, для начала выполним подключение библиотеки **itertools** и импортируем **product**, так как в данном типе задач допустимо любая комбинация символов алфавита:

**from** itertools **import** product

 Так как нам необходимо пройтись по всем возможным комбинациям, используем цикл **for**, а для того, чтобы указать длину последовательности, используем **repeat**:

**for** i **in** product("алфавит", **repeat** = длина последовательности):

 Для того, чтобы узнать сколько раз в последовательности встречается/не встречается тот или иной символ алфавита, используем count . И если требуемое условие выполнено, то увеличиваем счетчик на единицу:

Например, известно, что в записи числа цифра 3 должна встречаться 2 раза:

**if** i.count('3') == 2:

k +=1

print (k)

Если же в задании сказано, что на каком-либо определенном месте в числе должен стоять конкретная цифра, например, обязательное условие всегда должно быть то, что число не может начинаться с 0:

**if** i[0] != '0':

либо же, на третьем месте должна стоять цифра 3:

**if** i[2] == '3':

Также иногда требуется задать записать чисел слева направо в возрастающем или невозрастющем порядке. Например, в невозрастающем порядке это будет выглядеть следующим образом:

**if** i[0] >= i[1] >= i[2]:

Если же в условии сказано, что некоторые цифры должны или не должны стоять рядом, то необходимо из списка получить только одну строку, для можно использовать **join**. Например, цифра 1 не должна стоять рядом с 4:

**if** (i[0] != "0" **and** "14" **not in** "".join(i) **and** "41" **not** **in** "".join(i)):

Пример 23 (Демонстрационный вариант ЕГЭ 2023). Определите количество пятизначных чисел, записанных в восьмеричной системе счисления, в записи которых только одна цифра 6, при этом никакая нечётная цифра не стоит рядом с цифрой 6.

код для решения задания 8 демонстрационного варианта ЕГЭ 2023 будет выглядеть следующим образом:

**from** itertools **import** product

cnt = 0

**for** i **in** product ("01234567", **repeat** = 5):

**if** (i[0]!="0" **and** i.count("6")==1 **and**

    "16" **not in** "".join(i) **and**

    "36" **not** **in** "".join(i) **and**

    "56" **not in** "".join(i) **and**

    "76" **not in** "".join(i) **and**

    "61" **not in** "".join(i) **and**

    "63" **not in** "".join(i) **and**

    "65" **not in** "".join(i) **and**

    "67" **not in** "".join(i)):

        cnt += 1

print(cnt)

Ответ: 2961

Наиболее часто встречающийся тип задания:

Все *n*-буквенные слова, в составе которых могут быть только буквы А, Б, Г, Д...., записаны в алфавитном порядке и пронумерованы, начиная с 1. Ниже приведено начало списка.

1. ААААА
2. ААААБ
3. ААААВ
4. ААААГ
5. ААААД
6. АААБА

....

Под каким номером в списке идет первое/последнее слово, которое *содержит/не содержит.....*

При решении данного типа задания написание кода выполняется аналогично. Только теперь уже счетчик необходимо начать не с 0, а с единицы, так как мы считаем порядковые номера.

Пример 24 (Досрочный этап ЕГЭ 06.03.2023). Все четырехбуквенные слова, в составе которых могут быть только русские буквы А, В, Л, О, Р записаны в алфавитном порядке и пронумерованы начиная с 1. Ниже приведено начало списка.

1. АААА
2. АААВ
3. АААЛ
4. АААО
5. АААР
6. ААВА

.....

Под каким номером идет первое слово, начинающееся на Л?

Так как в данной задаче нам необходимо найти самое первое слово, удовлетворяющее условию задачи, то найдя его, нужно покинуть цикл, т.е., использовать **break**:

**from** itertools **import** product

cnt = 1

**for** i **in** product ('АВЛОР', **repeat** = 4):

**if** i[0]=='Л':

        print(n)

**break**

    cnt += 1

Пример 25 (Открытый банк заданий ФИПИ). Все шестибуквенные слова, составленные из букв М, У, Ж, Ч, И, Н, А, записаны в алфавитном порядке и пронумерованы.

Вот начало списка:

1.  АААААА

2.  АААААЖ

3.  АААААИ

4.  АААААМ

5.  АААААН

6.  АААААУ

7.  АААААЧ

……

Определите в этом списке количество слов с чётными номерами, которые не начинаются с буквы Ж и при этом содержат в своей записи не более одной буквы Ч.

Решение на Python:

**from** itertools **import** product

cnt1=0

cnt2=0

**for** i **in** product ('АЖИМНУЧ', **repeat** = 6):

cnt1 += 1 #выводим все комбинации

**if** i[0] != "Ж" **and** i.count('Ч') <= 1 **and** cnt1 % 2 == 0: #выбираем только с четными номерами, которые не начинаются с буквы Ж и при этом содержат в своей записи не более одной буквы Ч

cnt2+=1

print(cnt2)

Следующее задание, которое целесообразнее решать методом программирования – это номер 12. Данное задание не вызывает сильных сложностей у школьников, так как в нем, по сути, необходимо переписать данный в условии алгоритм, и проверить его результат.

Поэтому рассмотрим наиболее сложные прототипы.

Пример 26 (Открытый банк заданий ФИПИ). Исполнитель Редактор получает на вход строку цифр и преобразовывает её. Редактор может выполнять две команды, в обеих командах *v*и*w*обозначают цепочки цифр.

А) **заменить**(*v*,*w*).

Эта команда заменяет в строке первое слева вхождение цепочки *v*на цепочку *w.*Например, выполнение команды

**заменить**(111, 27)

преобразует строку 05111150 в строку 0527150.

Если в строке нет вхождений цепочки *v*, то выполнение команды

**заменить**(*v*,*w*) не меняет эту строку.

Б) **нашлось**(*v*).

Эта команда проверяет, встречается ли цепочка *v* в строке исполнителя Редактор. Если она встречается, то команда возвращает логическое значение «истина», в противном случае возвращает значение «ложь». Строка исполнителя при этом не изменяется.

Цикл

ПОКА  *условие*

*последовательность команд*

КОНЕЦ ПОКА

выполняется, пока условие истинно.

В конструкции

ЕСЛИ  *условие*

     ТО *команда1*

ИНАЧЕ *команда2*

КОНЕЦ ЕСЛИ

выполняется *команда1*(если условие истинно) или *команда2*(если условие ложно).

Дана программа для Редактора:

НАЧАЛО

ПОКА нашлось (52) ИЛИ нашлось (1122) ИЛИ нашлось (2222)

    ЕСЛИ нашлось (52)

      ТО заменить (52, 11)

    КОНЕЦ ЕСЛИ

    ЕСЛИ нашлось (2222)

       ТО заменить (2222, 5)

КОНЕЦ ЕСЛИ

    ЕСЛИ нашлось (1122)

       ТО заменить (1122, 25)

    КОНЕЦ ЕСЛИ

КОНЕЦ ПОКА

КОНЕЦ

На вход приведённой выше программе поступает строка, начинающаяся с цифры «5», а затем содержащая *n* цифр «2» (3 < *n <* 10 000).

Определите **наименьшее** значение *n*, при котором сумма цифр в строке, получившейся в результате выполнения программы, равна 64.

Решение на Python будет выглядеть следующим образом:

**for** n **in** range(3,10000):

s = '5' + '2' \* n #задаем строку, которая начинается с цифра «5», а затем содержащая n цифр «2».

**while** '52' **in** s **or** '1122' **in** s **or** '2222' **in s:**

**if** '52' **in** s**:**

s = s.replace('52','11',1)

**if** '2222' **in** s**:**

s = s.replace('2222','5',1)

**if** '1122' **in** s**:**

s = s.replace('1122','25',1)

**if sum**(**map**(**int,**s)) == 64:

print(n)

В данной программе замена символов производится с помощью функции **replace.** Например, запись s = s.replace('52','11',1) означает, что ‘52’ заменяется на ‘11’ один раз.

Также функция **map** применяет функцию **int** ко всем элементам итерируемого объекта **s**. Это означает, что каждый элемент **s** будет преобразован в целое число (**s** обычно представляет собой список строк или другой итерируемый объект, содержащий значения, которые нужно преобразовать в числа). Функция **sum** вычисляет сумму всех элементов, переданных ей в виде итерируемого объекта. В данном случае она суммирует все целые числа, полученные из **map(int, s)**.

В результате работы программы мы получим 3 числа, из которых необходимо выбрать наименьшее.

Ответ: 152.

Пример 27 (Открытый банк заданий ФИПИ). Исполнитель Редактор получает на вход строку символов и преобразовывает её.

Редактор может выполнять две команды, в обеих командах *v*и *w*обозначают цепочки символов.

**А) заменить**(*v*, *w*).

Эта команда заменяет в строке первое слева вхождение цепочки *v* на цепочку *w*.Например, выполнение команды

**заменить**(111, 27)

преобразует строку 05111150 в строку 0527150.

Если в строке нет вхождений цепочки *v*, то выполнение команды **заменить**(*v*, *w*) не меняет эту строку.

**Б) нашлось**(*v*).

Эта команда проверяет, встречается ли цепочка *v*в строке исполнителя Редактор. Если она встречается, то команда возвращает логическое значение «истина», в противном случае возвращает значение «ложь». Строка исполнителя при этом не изменяется.

Цикл

    ПОКА *условие*

*последовательность команд*

    КОНЕЦ ПОКА

выполняется, пока условие истинно.

В конструкции

    ЕСЛИ *условие*

        ТО *команда1*

    КОНЕЦ ЕСЛИ

выполняется *команда1*(если условие истинно).

В конструкции

    ЕСЛИ *условие*

        ТО *команда1*

        ИНАЧЕ *команда2*

    КОНЕЦ ЕСЛИ

выполняется *команда1*(если условие истинно) или *команда2*(если условие ложно).

На вход приведённой  ниже программы поступает строка, начинающаяся с символа «>»,  а затем содержащая  15 цифр 1, 20 цифр 2 и 16 цифр 3, расположенных в произвольном порядке.

Определите сумму числовых значений цифр строки, получившейся в результате выполнения программы.

Так, например, если результат работы программы представлял бы собой строку, состоящую из 50 цифр 4, то верным ответом было бы число 200.

НАЧАЛО

ПОКА  **нашлось** (>1)  ИЛИ **нашлось** (>2)  ИЛИ **нашлось** (>3)

         ЕСЛИ  **нашлось** (>1)

              ТО **заменить** (>1, 22>)

         КОНЕЦ ЕСЛИ

         ЕСЛИ  **нашлось** (>2)

              ТО **заменить** (>2, 2>)

         КОНЕЦ ЕСЛИ

         ЕСЛИ  **нашлось** (>3)

              ТО **заменить** (>3, 1>)

         КОНЕЦ ЕСЛИ

КОНЕЦ ПОКА

КОНЕЦ

Решая данную задачу, можно понять, что результатом работы данного алгоритма будет строка, состоящая из единиц и двоек, а таже символов «>». Поэтому посчитать сумму цифр с помощью **sum(map(int,s))** уже не получится, так как символ «>» не является цифрой.

Решение на Python будет выглядеть следующим образом:

**for** n **in** range(3,10000):

s = '>' + '1' \* 15 + '2' \* 20 + '3' \* 16

**while** '>1' **in** s **or** '>2' **in** s **or** '>3' **in** s:

**if** '>1' **in** s:

s = s.replace('>1', '22>', 1)

**if** '>2' **in** s:

s = s.replace('>2', '2>', 1)

**if** '>3' **in** s:

s = s.replace('>3', '1>', 1)

print(s.count('2') \* 2 + s.count('1'))

Таким образом, суммой цифр полученной строки будет количество двоек, умноженных на 2 и количество единиц.

Ответ: 116.

Рассмотрим методику решение нового задания № 13. Данное задание некоторым школьникам кажется достаточно сложным из-за слабых знаний основ﻿ построения﻿ компьютерных﻿ сетей. Данный тип заданий возможно также рашать «ручным» решением.

Пример 28 (Открытый банк заданий ФИПИ). В терминологии сетей TCP/IP маской сети называется двоичное число, определяющее, какая часть IP-адреса узла сети относится к адресу сети, а какая – к адресу самого узла в этой сети. Обычно маска записывается по тем же правилам, что и IP-адрес, – в виде четырёх байтов, причём каждый байт записывается в виде десятичного числа. При этом в маске сначала (в старших разрядах) стоят единицы, а затем с некоторого разряда – нули. Адрес сети получается в результате применения поразрядной конъюнкции к заданному IP-адресу узла и маске.

Например, если IP-адрес узла равен 231.32.255.131, а маска равна 255.255.240.0, то адрес сети равен 231.32.240.0.

Для узла с IP-адресом 117.191.176.37 адрес сети равен 117.191.160.0. Чему равен третий слева байт маски? Ответ запишите в виде десятичного числа.

Рассмотрим пример теоретического решения

Для узла с IP-адресом 117.191.176.37 адрес сети равен 117.191.160.0. Какой в этом случае может быть маска?

Так как адрес сети получается в результате применения поразрядной конъюнкции к IP-адресу узла и маске, то мы можем записать:

|  |  |  |
| --- | --- | --- |
| IP-адрес узла: | 117.191.176. 37 | 117.191.176. 37 |
| Маска: | & *xxx. xxx. xxx. xxx* ⇒ | & 255*.* 255*. xxx.* 0 |
| Адрес сети: | 117.191. 160. 0 | 117.191. 160. 0 |

Мы видим, что 1-й и 2-й байты маски – единичные, а 4-й байт – нулевой. Будем «реконструировать» 3-й байт маски.

17610 = 101100002,

19210 = 101000002.

Если 3-й байт маски представить в виде *хххххххх*2, то можно записать:

|  |
| --- |
| 1 0 1 1 0 0 0 0 |
| & *x x x x x x x x* |
| 1 0 1 0 0 0 0 0 |

Первая, третья и четвертая слева цифры, принадлежащие рассматриваемому байту маски, определяются однозначно и равны соответственно 1, 1 и 0:

|  |
| --- |
| 1 0 1 1 0 0 0 0 |
| & 1 *x* 10 *x x x x* |
| 1 0 1 0 0 0 0 0 |

Из того, что маска – 32-битное число, в двоичной записи которого сначала стоят единицы, а потом – нули, следует, что после нуля, стоящего на четвёртом месте, могут следовать только нули:

|  |
| --- |
| 1 0 1 1 0 0 0 0 |
| & 11100 0 0 0 |
| 1 0 1 0 0 0 0 0 |

Тогда для третьего байта маски возможно следующее значение:

111000002 = 22410

Рассмотрим решение с помощью программирования:

**from** ipaddress **import** \*

ip\_add = ip\_address("117.191.176.37")

**for** mask **in** range(33):

**try**:

ip\_net = ip\_network(f"117.191.160.0/{mask}")

**if** ip\_add **in** ip\_net:

print(ip\_net.netmask)

**except**:

**continue**

Данный код на Python использует библиотеку **ipaddress** для работы с IP-адресами и сетями. Давайте разберем его по частям:

1. Импорт библиотеки:

**from** ipaddress **import**\* – импортирует все классы и функции из модуля **ipaddress**, который предоставляет инструменты для создания, манипуляции и анализа IP-адресов и сетей.

1. Создание IP-адреса:

ip\_add = ip\_address("117.191.176.37") – создает объект IP-адреса с адресом 117.191.176.37.

1. Цикл по маскам:

**for** mask **in** range(33): – создает цикл, который перебирает значения от 0 до 32 (включительно). Эти значения представляют собой возможные длины маски подсети в битах (например, 0 – это маска 0.0.0.0, а 32 – это маска 255.255.255.255).

1. Создание сети:

ip\_net = ip\_network(f"117.191.160.0/{mask}") – для каждой итерации цикла создается объект сети с базовым адресом 117.191.160.0 и текущей длиной маски.

1. Проверка принадлежности IP-адреса к сети:

**if** ip\_add **in** ip\_net: – проверяет, принадлежит ли IP-адрес 117.191.176.37 к текущей сети ip\_net.

1. Вывод маски сети:

print(ip\_net.netmask) – если IP-адрес принадлежит сети, выводится маска подсети для этой сети.

1. Обработка исключений:

except:

continue – если возникает ошибка при создании сети (например, если маска недопустима), программа просто продолжает выполнение следующей итерации цикла.

В результате работы программы мы получим саму маску, и увидим, что третий байт маски равен 224. (см. рис. 6).

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Рисунок 6 – Результат работы программы

Пример 29 (Демонстрационный вариант ЕГЭ 2025) В терминологии сетей TCP/IP маской сети называется двоичное число, определяющее, какая часть IP-адреса узла сети относится к адресу сети, а какая – к адресу самого узла в этой сети. Адрес сети получается в результате применения поразрядной конъюнкции к заданному IP-адресу узла и маске сети.

Сеть задана IP-адресом 172.16.168.0 и маской сети 255.255.248.0.

Сколько в этой сети IP-адресов, для которых количество единиц в двоичной записи IP-адреса **не кратно** 5?

Ответ запишите в виде десятичного числа.

**from** ipaddress **import** \*

ip\_net = ip\_network('172.16.168.0/255.255.248.0') #создание объекта сети

cnt = 0

**for** ip\_add **in** ip\_net: # перебор IP-адресов в сети

**if bin**(**int**(ip\_add)).count('1') % 5 != 0:

cnt += 1

print(cnt)

Рассмотрим условие для подсчета:

**if bin**(**int**(ip\_add)).count('1') % 5 != 0:

cnt += 1

* **int**(ip\_add) – преобразует текущий IP-адрес в целое число.
* **bin**(...) – преобразует это целое число в двоичную строку.
* count('1') – подсчитывает количество единиц в двоичной строке.
* **if** ... % 5 != 0 – проверяет, делится ли количество единиц на 5 без остатка. Если не делится, счетчик **cnt** увеличивается на 1.

После завершения цикла программа выводит значение счетчика cnt, которое представляет собой количество IP-адресов в сети, у которых количество единиц в двоичном представлении не делится на 5.

Сеть 172.16.168.0/255.255.248.0 имеет 2048 адресов (от 172.16.168.0 до 172.16.175.255), и в результате выполнения кода будет подсчитано, сколько из этих адресов соответствуют заданному условию.

Ответ: 1663

Теоретически данную задачу решить сложнее, так как у нас 2048 адресов, которые нужно перевести в двоичную систему счисления, после чего посчитать количество единиц в каждой двоичной записи и проверить их делимость на 5.

Таким образом, использование программирования на Python для решения некоторых заданий ЕГЭ по информатике, которые не направлены на проверку умения писать программный код, не только облегчает процесс, но и способствует развитию навыков программирования в целом, которые необходимы для решения сложных задач ЕГЭ по информатике, таких как номера 24–27.