Министерство общего и профессионального образования

Свердловской области

Управление образования Администрации города Нижний Тагил

Муниципальное автономное общеобразовательное учреждение

лицей № 39

Направление: научно-техническое

Секция: математика и информатика

**Применение языка программирования Python в решении задач компьютерного ЕГЭ по информатике**

Исполнитель: учащийся 11А класса

Губаев Максим Станиславович

Руководитель: Гребнева Дарья Михайловна, учитель информатики

Нижний Тагил

2020

**Оглавление**

[Введение 3](#_Toc31874439)

[1. Основы языка программирования Python 4](#_Toc31874440)

[2. Обзор задач компьютерного ЕГЭ по информатике и их решение на языке Python 7](#_Toc31874441)

[3. Сравнение эффективности программ, написанных на языках Pascal, C и Python 16](#_Toc31874442)

[Заключение 18](#_Toc31874443)

## Введение

Единый государственный экзамен по информатике необходим тем, кто планирует поступать в российские вузы на специальности, связанные с IT-технологиями. Этот экзамен нужен тем, кто хочет стать программистом, разработчиком, специалистом по информационным технологиям.

Единый государственный экзамен по информатике будет проходить на компьютерах уже с 2021 года. Новая модель реализована в виде компьютерной системы тестирования, а ее апробация прошла в нашем городе осенью 2019. Смысл новой модели состоит в том, что все задания выпускники будут выполнять при помощи компьютеров и с применением различных языков программирования и программного обеспечения.

В настоящее время все большую популярность приобретает язык Python. Одна из причин популярности Python  – более простое и компактное оформление, чем в других языках. Это самый популярный язык общего назначения: он используется для машинного обучения, аналитике, разработке игр и в науке о данных. В данной работе будет применение языка Python в решении задач компьютерного ЕГЭ по информатике.

**Объект работы** – процесс решения задач компьютерного ЕГЭ по информатике.

**Предмет работы** – средства решения задач компьютерного ЕГЭ по информатике.

**Цель работы** – провести обзор возможностей языка программирования Python в решении задач компьютерного ЕГЭ по информатике.

**Задачи:**

* рассмотреть основы языка программирования Python;
* выделить типы задач компьютерного ЕГЭ по информатике и, по возможности, решить их средствами языка программирования Python;
* сравнить эффективность программ, написанных на языках Pascal, C и Python.

## 1. Основы языка программирования Python

Сценарии исходного кода Python состоят из так называемых *логических строк*, каждая из которых в свою очередь состоит из *физических строк*. Для обозначения комментариев используется символ #. Комментарии и пустые строки интерпретатор игнорирует.

Физические строки разделяются самим символом конца строки. Для выделения блоков кода используются исключительно отступы. Логические строки с одинаковым размером отступа формируют блок, и заканчивается блок в том случае, когда появляется логическая строка с отступом меньшего размера. Именно поэтому первая строка в сценарии Python не должна иметь отступа.

Других радикальных отличий от других языков программирования в синтаксисе Python нет. Также используются стандартные правила для заданий идентификаторов переменных, методов и классов – имя должно начинаться с подчеркивания или латинского символа любого регистра и не может содержать символов @, $, %. Также не может использоваться в качестве идентификатора только один символ подчеркивания.

Типы данных, используемых в Python, совпадают с другими языками – целые и вещественные типы данных; дополнительно поддерживается комплексный тип данных – с вещественной и мнимой частью. Python поддерживает строки, которые могут быть заключены в одинарные, двойные или тройные кавычки, при этом строки являются immutable-объектами, т.е. не могут изменять свое значение после создания.

Есть в Python и логический тип данных bool c двумя вариантами значения – True и False. Для повышения читаемости кода рекомендуется использовать для логических переменных тип bool.

В Python определены три типа коллекций для хранения наборов данных:

* кортеж (tuple);
* список (list);
* словарь (dictionary).

Кортеж представляет собой неизменяемую упорядоченную последовательность данных. В нем могут содержаться элементы различных типов, например другие кортежи. Кортеж определяется в круглых скобках, а его элементы разделяются запятыми. Специальная встроенная функция tuple() позволяет создавать кортежи из представленной последовательности данных.

Список – это изменяемая упорядоченная последовательность элементов. Элементы списка также разделяются запятыми, но задаются уже в квадратных скобках. Для создания списков предлагается функция list().

Словарь является хеш-таблицей, сохраняющей элемент вместе с его идентификатором-ключом. Последующий доступ к элементам выполняется тоже по ключу, поэтому единица хранения в словаре – это пара объект-ключ и связанный с ним объект-значение. Словарь – это изменяемая, но не упорядоченная коллекция, так что порядок элементов в словаре может меняться со временем. Задается словарь в фигурных скобках, ключ отделяется от значения двоеточием, а сами пары ключ/значение разделяются запятыми. Для создания словарей доступна функция dict().

В листинге 1 приведены примеры различных коллекций, доступных в Python.

Листинг 1. Виды коллекций, доступные в Python

|  |  |
| --- | --- |
|  | (‘w’,‘o’,‘r’,‘l’,‘d’)     # кортеж из пяти элементов  (2.62,)                   # кортеж из одного элемента  [“test”,'me']             # список из двух элементов  [  ]                      # пустой список  { 5:‘a’, 6:‘b’, 7:‘c’ }   # словарь из трех элементов с ключами типа int |

Многие возможности Python реализованы в виде отдельных функций; кроме того, модули расширения чаще всего делаются тоже в виде библиотеки функций. Функции также применяются и в классах, где они по традиции называются методами.

Синтаксис определения функций в Python крайне простой; с учетом изложенных выше требований (листинг 2).

Листинг 2. Виды коллекций, доступные в Python

|  |  |
| --- | --- |
|  | def ИМЯ\_ФУНКЦИИ(параметры):      выражение № 1      выражение № 2      ... |

Как видно, необходимо использовать служебное слово def, двоеточие и отступы. Вызвать функцию также очень просто:

|  |  |
| --- | --- |
|  | ИМЯ\_ФУНКЦИИ(параметры) |

Есть только несколько моментов, специфичных для Python, которые стоит учитывать. Параметры могут передаваться как просто по порядку перечисления, так и по именам, в этом случае не нужно указывать при вызове те параметры, для которых есть значения по умолчанию, а передавать только обязательные или менять порядок параметров при вызове функции (листинг 3).

Листинг 3. Виды коллекций, доступные в Python

|  |  |
| --- | --- |
|  | #функция, выполняющая деление нацело – с помощью оператора //  def foo(delimoe, delitel):      return delimoe // delitel    print divide(50,5)                         # результат работы: 10  print divide(delitel=5, delimoe=50)      # результат работы: 10 |

Функция в Python обязательно возвращает значение – это делается либо явно с помощью оператора return, за которым следует возвращаемое значение, либо, в случае отсутствия оператора return, возвращается константа None, когда достигается конец функции. Как видно из примеров объявлений функций, в Python нет необходимости указывать, возвращается что-либо из функции или нет, однако если в функции имеется один оператор return, возвращающей значение, то и другие операторы return в этой функции должны возвращать значения, а если такого значения нет, то необходимо явно прописывать return None.

Если функция очень простая и состоит из одной строки, то ее можно определить прямо на месте использования, в Python подобная конструкция называется лямбда-функцией (lambda). lambda-функция – это анонимная функция (без собственного имени), телом которой является оператор return, возвращающий значение некоторого выражения. Такой подход может оказаться удобным в некоторых ситуациях, однако стоит заметить, что повторное использование подобных функций невозможно.

Еще стоит описать отношение Python к использованию рекурсии. По умолчанию глубина рекурсии ограничена 1000 уровней, и когда этот уровень будет пройден, возникнет исключительная ситуация, и работа программы будет остановлена. Однако при необходимости величину этого предела можно изменить.

## 2. Обзор задач компьютерного ЕГЭ по информатике и их решение на языке Python

В проекте компьютерного ЕГЭ по информатике предлагаются десять типов заданий на следующие темы.

1. Вычисления
2. Решение уравнений численными методами
3. Перебор целых чисел. (Разбиение числа на цифры)
4. Перебор чисел. Проверка делимости
5. Перебор целых чисел. Количество делителей
6. Символьные строки. Цепочки символов
7. Функции двух аргументов. Таблицы значений
8. Электронные таблицы. Встроенные функции (не решается средства Python)
9. Рекурсия. Рекурсивные функции
10. Исследование моделей. Оптимизация
11. *Пример задания на вычисление*

|  |
| --- |
| С помощью программы Калькулятор или электронных таблиц вычислите значение выражения. В ответе запишите только целую часть результата. Можно также написать программу. |

Программа на языке Python

**from math import sqrt, cos, pi**

**print( sqrt(1 + cos(3.53\*pi)\*10)\*310 )**

Ответ: 431.

Для решения данного задания, нужно знать правила записи математических функций на языке Python. В связи с невозможностью записи некоторых стандартных математических функций с клавиатуры персонального компьютера в языке Python существуют так называемые встроенные функции, с помощью которых пользователь записывает арифметические выражения.

Основные математические функции языка Python представлены в таблице 1. Прежде чем использовать математические функции, необходимо в начале программы написать инструкцию import math, однако тогда перед упоминанием каждой функции необходимо будет добавлять имя модуля - math, например, y=math.sin(x). Другой способ, который позволит избежать многократного вызова модуля math, - сделать следующую запись в начале программы: from math import \*.

Таблица 1. Общие математические функции модуля Math

|  |  |
| --- | --- |
| **Запись на Python** | **Действие** |
| math.sin (x) | Возвращает значение функции Sin от числа х |
| math.cos (x) | Возвращает значение функции Cos от числа х |
| math.tan (x) или math.sin (x)  / math.cos (x) | Возвращает значение функции Tg от числа х |
| math.cos (x) / math.sin (x) | Возвращает значение функции Ctg от числа х |
| math.abs (x) | Возвращает абсолютную величину числа х |
| math.exp (x) | Возвращает результат возведения числа е в степень X |
| math.Log lp (x) | Возвращает натуральный логарифм от х+1 |
| math.sqrt (x) | Возвращает результат извлечения квадратного корня числа х |
| math.log (x) | Возвращает логарифм числа х по основанию  10 |
| math.cos (x) \* math.cos (x) | Возвращает результат возведения функции  Cos х в квадрат |
| math.acos (x) | Возвращает значение функции арккосинус от числа х |
| math.asin (x) | Возвращает значение функции арксинус от числа х |
| math.atan (x) | Возвращает значение функции арктангенс от числа х |
| Pi | Возвращает 3.141592653589793 |
| math.degrees(x) | Преобразует радианы в градусы |
| math.radians(x) | Преобразует градусы в радианы |
| math.floor(x) | Возвращает значение, округленное до ближайшего меньшего целого |
| math.ccil(x) | Возвращает значение, округленное до ближайшего большего целого |
| math.factorial(x) | Возвращает факториал числа. 3 != 1 \*2\*3 |

В таблице 2 представлены некоторые встроенные функции для работы с числами, не требующие подключения модуля math.

Таблица 2. Функции для работы с числами

|  |  |
| --- | --- |
| **Запись на Python** | **Описание** |
| round(x) | Возвращает результат округления числа х до ближайшего меньшего целого значения для чисел с дробной частью меньше 0.5 или результат округления числа х до ближайшего большего целого значения для чисел с дробной частью больше 0.5 |
| pow(x,y)  другой вариант х\*\*у | Возвращает результат возведения числа х в степень у |
| mах(список чисел через запятую) | Возвращает большее значение из списка чисел |
| min(список чисел через запятую) | Возвращает меньшее значение из списка чисел |
| sum(список K чисел через запятую) | Возвращает сумму значений элементов последовательности |
| float(число) | Преобразует объект (например, строковое значение, целое значение) в вещественное число |

*2. Пример задания на решение уравнения численным методом*

|  |
| --- |
| Известно, что уравнение  на отрезке [0; 1,5] имеет единственный корень. Найдите его приблизительное значение с точностью не менее 0,00001 и запишите в ответе найденное значение ровно с пятью значащими цифрами после запятой. |

Программа на языке Python:

**from math import cos, exp # подключить функции cos, exp**

**def f(x): # это функция f(x)**

**return 0.01\*exp(x) - cos(3\*x)**

**a, b = 0, 1.5 # границы отрезка**

**while b-a > 1e-6: # пока ширина отрезка >= 10^(-6)**

**c = (a + b) / 2 # середина отрезка**

**if f(a)\*f(c) <= 0: # сдвигаем правую или левую границу**

**b = c**

**else: a = c**

**# вывод с 5 знаками в дробной части**

**print( "{:.5f}".format((a + b) / 2) )**

Ответ: 0.51800

*3. Пример задания на перебор целых чисел. Разбиение числа на цифры*

|  |
| --- |
| *Назовём натуральное четырёхзначное число N (1000 ≤ N ≤ 9999) счастливым, если суммы двух его первых и двух последних цифр различаются не более, чем на 3. Найдите количество таких чисел.* |

Программа на языке Python

**count = 0**

**for n in range(1000, 10000):**

**d0 = n % 10; n //= 10**

**d1 = n % 10; n //= 10**

**d2 = n % 10**

**d3 = n // 10**

**if abs(d3+d2-d1-d0) <= 3:**

**count += 1**

**print(count)**

Поскольку заданный отрезок [1000; 9999] содержит всего 9000 чисел, можно решать задачу простым перебором. Для этого сначала нужно разбить число на цифры с помощью операций деления нацело и остатка от деления; цифры помещаем в переменные d0, d1, d2, d3. Затем проверяем «счастливость» числа: число счастливое при выполнении условия в этом случае увеличиваем счётчик найденных счастливых чисел.

Ответ: 4071.

*4. Пример задания на перебор целых чисел. Проверка делимости*

|  |
| --- |
| *Рассматривается множество целых чисел, принадлежащих отрезку* [1033; 7737], которые делятся на 5 и не делятся на 11, 17, 19 и 23. *Найдите количество таких чисел и максимальное из них. В ответе запишите два числа через пробел: сначала количество, затем максимальное число.* |

Программа на языке Python

**count = 0**

**maxGood = 0**

**for n in range(1033, 7737+1):**

**if (n % 5 == 0) and (n % 11 != 0) and \**

**(n % 17 != 0) and (n % 19 != 0) and (n % 23 != 0):**

**maxGood = n**

**count += 1**

**print(count, maxGood)**

Поскольку заданный отрезок [1033; 7737] содержит не так много чисел, можно решать задачу простым перебором. Условие будем понимать так: интересующие нас числа делятся на 5 и не делятся ни на одно из чисел 11, 17, 19 и 23. Нам выгоднее перебирать числа в порядке возрастания, тогда последнее найдённое число – это и есть искомое максимальное подходящее число (если требуется найти наименьшее подходящее число, удобнее перебирать числа в порядке убывания)

Ответ: 1040 7730

|  |
| --- |
| *Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку* [194455; 194500], *числа, имеющие ровно 4 различных делителя. Выведите эти четыре делителя для каждого найденного числа в порядке возрастания.* |

Программа на языке Python

**for n in range(194455, 194500+1):**

**divs = []**

**for d in range(1,n+1):**

**if n % d == 0:**

**divs.append(d)**

**if len(divs) == 4:**

**print( \*divs )**

При написании программы на языке Python можно поступить так

**for для всех чисел n в интервале:**

**divs = массив всех делителей n**

**if len(divs) == 4:**

**вывести массив делителей**

*5. Пример задания на работу с простыми числами*

|  |
| --- |
| *Напишите программу, которая ищет среди целых чисел, принадлежащих числовому отрезку* [3532000; 3532160], *простые числа. Выведите все найденные простые числа в порядке возрастания, слева от каждого числа выведите его номер по порядку.* |

**Программа на языке Python**

**from math import sqrt**

**count = 0**

**for n in range(3532000, 3532160+1):**

**prime = True**

**for d in range(2, round(sqrt(n))):**

**if n % d == 0:**

**prime = False**

**break**

**if prime:**

**count += 1**

**print( count, n )**

*6. Пример задания на работу с символьными строками*

|  |
| --- |
| В текстовом файле k7.txt находится цепочка из символов латинского алфавита A, B, C, D, E. Найдите количество цепочек длины 3, удовлетворяющих следующим условиям:   * 1-й символ – один из символов B, C или D; * 2-й символ – один из символов B, D, E, который не совпадает с первым; * 3-й символ – один из символов B, C, E, который не совпадает со вторым. |

Программа на языке Python

**s = open('k7.txt').read()**

**count = 0**

**for i in range(len(s)-2):**

**if s[i] in 'BCD' and s[i+1] in 'BDE' \**

**and s[i+2] in 'BCE' and s[i]!=s[i+1] \**

**and s[i+1]!=s[i+2]:**

**count += 1**

**print(count)**

**Решение:**

1. Считываем из файла и перебираем символы.
2. Перебираем все тройки символов. Примем, что переменная i будет хранить номер первого элемента в тройке, то есть, будем рассматривать тройки (**s[i]**, **s[i+1]**, **s[i+2]**).
3. Организуем цикл который перебирает значения **i** от 1 до **len(s)-2**

**for i in range(len(s)-2):**

**...**

1. Проверяем символы в каждой тройке на соответствие условию. Проверка принадлежности символов набору аналогична заданию 1. Дополнительно необходимо указать условия неравенства символов, указанных в условии задачи. Если условия выполняются, то к переменной количества прибавляется единица.

*7. Пример задания на вычисление значения функции от двух переменных*

|  |
| --- |
| С помощью редактора электронных таблиц создайте таблицу вещественных значений выражения  для следующих вещественных значений x и y:  x = 5,5; 6,0; …; 8,5; y = 10,0; 10,3; …; 13,0.  Вычислите сумму получившихся значений и запишите её целую часть в ответе.  Для выполнения этого заданий также можно написать программу. |

**Решение:**

1. Чтобы написать программу, нужно использовать вложенный цикл: в одном цикле будем перебирать значения *x*, а во втором (вложенном) – значения *y*; учитывая, что цикл с переменной (**for ... in ...**) работает только с целыми последовательностями чисел, придётся использовать циклы с условием:

**s = 0 # это неправильная программа**

**x = 5.5 # это неправильная программа**

**while x <= 8.5: # это неправильная программа**

**y = 10**

**while y <= 13:**

**# print(x, y) # отладочная печать, см. обсуждение ниже**

**s += 2\*x\*\*3/(y+1)**

**y += 0.3**

**x += 0.5**

**print( s )**

сумма значений функции накапливается в переменной **s**

1. однако эта программа выводит неверный ответ.
2. Дело в том, что вещественные числа, которые нельзя представить в виде суммы целых (в том числе и отрицательных) степеней числа 2, в двоичной системе счисления представляют собой бесконечную дробь и поэтому не могут быть точно записаны в памяти двоичного компьютера; при выполнении вычислений с такими числами ошибка накапливается, и к последнему шагу (это можно проверить с помощью отладочной печати) значение *y* равно не 12,7, а чуть больше:

**12.700000000000006**

из-за этого следующее значение, равное 13,000000000000006, уже больше, чем 13, и не удовлетворяет условию работы цикла; таким образом, на каждом шаге цикла по *x* мы теряем одно значение *y*, и соответствующее значение функции не включается в сумму.

1. с переменной *x* подобных проблем нет, так как шаг изменения *x* равен 0,5 = 2–1
2. исправить ситуацию можно так: организовать перебор только целых значений, используя вспомогательные целочисленные переменные *x*10 = *x* ⋅ 10 и *y*10 = *y* ⋅ 10:

**s = 0**

**for x10 in range(55, 86, 5):**

**for y10 in range(100, 131, 3):**

**s += 2\*(x10/10)\*\*3/(y10/10+1)**

**print(s)**

*8. Пример задания на вычисление значения рекурсивной функции*

|  |
| --- |
| Определите наименьшее значение n, при котором сумма чисел, которые будут выведены при вызове F(n), будет больше 500000. Запишите в ответе сначала найденное значение n, а затем через пробел – соответствующую сумму выведенных чисел.  **def F( n ):**  **print(2\*n)**  **if n > 1:**  **print(n-5)**  **F(n-1)**  **F(n-2)** |

**Решение:**

Первое, что может прийти в голову – вызывать приведённую процедуру при разных значениях параметра и увеличивать это значение до тех пор, пока сумма выведенных чисел не превысит заданное значение 500000; это тупиковый подход, поскольку чисел очень много и сложение займет очень много времени при низкой вероятности правильного ответа

Можно попробовать изменить программу так, чтобы сумма выводимых чисел считалась автоматически: добавим в программу глобальную переменную **s** и будем увеличивать её при выводе каждого числа на значение этого числа; при этом для ускорения (значительного!) работы программы сразу закомментируем вывод чисел на экран:

**def F( n ):**

**global s # если не объявить s глобальной – ошибка!**

**# print(2\*n)**

**s += 2\*n**

**if n > 1:**

**# print(n-5)**

**s += n - 5**

**F(n-1)**

**F(n-2)**

Дальше можно написать такую программу и запускать её при различных значениях переменной **n**:

**n = 15**

**s = 0**

**F(n)**

**print( n, s )**

Увеличивая каждый раз значение *n* на 1, мы в конце концов найдём первое (минимальное) значение *n*, при котором сумма чисел, которые будут выведены при вызове *F*(*n*), будет больше 500000 – это *F*(24) = 531864

Ответ: 24 531864.

*9. Пример задания на оптимизацию*

|  |
| --- |
| На покупку мебели выделено 500 тыс. рублей. Стоимость одного комплекта составляет 18 тыс. рублей. Запишите наборы вариантов покупки максимального количества комплектов мебели, при условии, что производитель М продает мебель упаковками по 6 комплектов в упаковке, а производитель N – по 4 комплекта в упаковке.  Запишите в ответ пары чисел: количество упаковок производителя М далее через пробел количество упаковок производителя N. Каждую пару записывайте с новой строки. Пары должны быть отсортированы по возрастанию значений в первом столбце. |

**Решение:**

В простейшем варианте можно просто вывести на экран все варианты сочетаний *a* и *b* с соответствующими значениями *K*, в конце программы вывести макcимальное значение *K*; затем вручную найти все строки, где значение *K* равно максимальному.

**S0 = 500000 # доступная сумма**

**cost1 = 18000 # стоимость одного комплекта**

**packM = 6 # количество комплектов в упаковке M**

**packN = 4 # количество комплектов в упаковке N**

**# максимальное значение a**

**aMax = int(S0 / (packM\*cost1))**

**# поиск максимального K по всем вариантам**

**maxK = 0**

**for a in range(aMax+1):**

**Sb = S0 - a\*packM\*cost1 # сумма на закупку у N**

**b = int(Sb / (packN\*cost1))**

**K = packM\*a + packN\*b # общее количество**

**print(a, b, K)**

**if K > maxK:**

**maxK = K # новый максимум**

**print(maxK, maxK\*cost1)**

## 3. Сравнение эффективности программ, написанных на языках Pascal, C и Python

Сравним программы, написанные на языках Pascal, C и Python по таким критериям, как время работы и используемая память. Для примера возьмем линейные программы, программы с циклами и программы с рекурсивными функциями.

*Задача 1:* дано натуральное число. Выведите его последнюю цифру.

|  |  |  |
| --- | --- | --- |
| Pascal | C | Python |
| var a: integer;  begin  readln(a);  writeln(a mod 10);  end. | #include <stdio.h>  main() {  int x;  scanf("%i", &x);  printf("%i", x%10);  } | x=int(input())  print(x%10) |
| Время работы: 0.002 с.  Используемая память: 348160 бит | Время работы: 0.002  Используемая память: 958464 | Время работы: 0.037  Используемая память:  360448 бит |

Как видно из таблицы, при реализации линейных алгоритмов программа на Python проигрывает во времени реализации и используемой памяти программам на Pascal и C.

*Задача 2:* подсчитайте количество натуральных делителей числа *x* (включая 1 и само число; x2109).

|  |  |  |
| --- | --- | --- |
| **Pascal** | **C** | **Python** |
| **var** x, i, k: longint;  **begin**  readln(x);  k:=0;  **for** i:=1 **to** x **do**  **begin**  **if** (x **mod** i =0) **then** inc(k);  **end**;  writeln(k);  **end**. | #include <stdio.h>  #include <conio.h>    main() {  int x, i, k;  scanf("%i", &x);  k=0;  for (i=1; i<=x; i++) {  if (x % i ==0) {  k=k+1;  }  }  printf("%i", k);  } | x=int(input())  k=0  for i in range(1,x+1):  if x%i==0:  k+=1  print(k) |
| Время работы: 0.002 с.  Используемая память: 348160 бит | Время работы: 0.002  Используемая память: 958464 | Время работы: 0.039  Используемая память:  376832 бит |

Из-за того, что значение x может быть достаточно большим, все написанные программы не проходят все тесты. Последние два теста не выполняются из-за того, что превышено максимальное время работы (рис. 1).



Рис. 1. Результаты прохождения тестов

Поэтому простым перебором данную задачу решать нельзя. Повысим эффективность программы, идея состоит в том, чтобы для определения количества делителей числа N перебирать только числа до ; если число q целое, его нужно добавить в список делителей, а все остальные делители – парные, то есть если a – делитель N, то b = N / a – тоже делитель N.

Таким образом, циклом пробегаемся от 1 до (корня X)-1, и проверяем на делимость, если делится, то увеличиваем счетчик. После цикла удваиваем наш счетчик, так как любое число если имеет делитель до корня этого числа, то он имеет ещё один делитель после корня этого числа. Проверяем отдельным if-ом случай для корня X, если делится нацело, то увеличиваем на один и выводим наш счетчик, иначе просто выводим счетчик.

|  |  |  |
| --- | --- | --- |
| **Pascal** | **C** | **Python** |
| **var** x, i, k: longint;  **begin**  readln(x);  k:=0;  **for** i:=1 **to** x **do**  **begin**  **if** (x **mod** i =0) **then** inc(k);  **end**;  writeln(k);  **end**. | #include <stdio.h>  #include <math.h>>  main() {  int x, i, k,q;  double fp,ip;  scanf("%i", &x);  k=0;  if (x%2==0) {  q=trunc(sqrt(x));  } else  {  q=trunc(sqrt(x))-1;  }  for (i=1; i<=q; i++) {  if (x % i ==0) {  k=k+1;  }  }  k=k\*2;  fp = modf(sqrt(x) , &ip);  if(fp==0) {  k=k+1;  }  printf("%i", k);  } | import math  x=int(input())  if (x%2==0):  q=math.trunc(math.sqrt(x))  else:  q=math.trunc(math.sqrt(x))-1  k=0  for i in range(1,q+1):  if (x%i==0):  k=k+1  k=k\*2  if (math.sqrt(x)%1==0):  k=k+1  print(k) |
| Время работы: 0.002 с.  Используемая память: 348160 бит | Время работы: 0.002  Используемая память: 962560 бит | Время работы: 0.037  Используемая память:  380928 бит |

## Заключение

Девяносто процентов задач компьютерного ЕГЭ по информатике решается с помощью программирования.

В процессе выполнения работы были решены следующие задачи:

* рассмотрены основы языка программирования Python;
* выделены типы задач компьютерного ЕГЭ по информатике. Задачи решены средствами языка программирования Python;
* проанализирована эффективность программ, написанных на языках Pascal, C и Python.

## Список литературы

## Сайт К. Ю. Полякова. Методические материалы и программное обеспечение [Электронный ресурс]. Режим доступа: http://kpolyakov.spb.ru

## Прохоренок Н. А. Python 3. Самое необходимое [Текст]. – Спб.: БХВ-Петербург, 2019. – 608 с.

1. Основы Python [Электронный ресурс]. Режим доступа: https://pythonworld.ru/osnovy/skachat-python.html
2. Применение и основы Python [Электронный ресурс]. Режим доступа: https://stepik.org/course/512/promo
3. Дистанционная подготовка к информатике [Электронный ресурс]. Режим доступа https://informatics.mccme.ru