муниципальное бюджетное общеобразовательное учреждение «Школа № 46 имени прокурора Куйбышевской области Н.А. Баженова» городского округа Самара

Индивидуальный итоговый проект

Предмет: «Информатика»

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

**Выполнил** ученик 10 Б класса

Швалёв Данила Александрович

**Руководитель проекта:**

учитель информатики

Конопа Галина Павловна

Самара

2024 г.

**Оглавление**

[Введение 3](#_Toc31874439)

[1. Язык программирования Python 4](#_Toc31874440)

[2. Решение задач по информатике на языке Python 7](#_Toc31874441)

[3. Сравнение эффективности программ, написанных на других языках с языком Python. 16](#_Toc31874442)

[Заключение 19](#_Toc31874443)

## Введение

**Актуальность моей работы** состоит в том, что в настоящее время все большую популярность приобретает язык Python, который используется во всём мире, в таких компаниях как Яндекс, Google и к тому же на данном языке создаётся множество игр.

Одна из причин популярности Python – более простое и компактное оформление, чем в других языках.

В данной работе я использую язык Python в решении задач по информатике.

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

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

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

**Задачи:**

* рассмотреть основы языка программирования Python;
* выделить типы задач по информатике, применяемых на экзамене и по возможности, решить их средствами языка программирования Python;
* сравнить эффективность программ, написанных на других языках с программой Python.

## Язык программирования Python

## Python (произносится как «пайтон», но многие говорят «питон») – это язык программирования общего назначения, который широко применяется в различных областях: от создания банальных веб-страниц до систем управления роверами на других планетах.

## Язык скриптовый, он универсален и является самым популярным языком программирования в мире (по данным издания Tiobe, составляющего топы наиболее востребованных языков).

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. Решение задач по информатике на языке Python

В проекте экзамена 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

Рассмотрим несколько заданий и решим их с помощью языка программирования Python.

**Пример задания на построение таблиц истинности логических выражений; строки с пропущенными значениями.**

Мы не всегда помним таблицу истинности, поэтому проще написать программу.

Задание 2

Логическая функция *F* задаётся выражением *(x ∨ y) → (z ≡ x)*.

Дан частично заполненный фрагмент, содержащий неповторяющиеся строки таблицы истинности функции *F*.Определите, какому столбцу таблицы истинности соответствует каждая из переменных *x*, *y*, *z*.

В ответе напишите буквы *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.

**Решение:**

print ('x y z')  
for x in range(0, 2):  
 for y in range(0, 2):  
 for z in range(0, 2):  
 if not((x==z) or (x<=(y and z))):  
 print(x, y, z)

And коньюнкция

Инверсия not

Or дизьюнкция

= = эквиваленция

< = импликация

**Пример задания на анализ и построение алгоритмов для исполнителей: посимвольное двоичное преобразование/посимвольное десятичное преобразование.**

**Задание 5**

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 цифр на 2.

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

Укажите такое наименьшее число *N*, для которого результат работы алгоритма больше числа 77. В ответе это число запишите в десятичной системе счисления.

**Решение:**

def f(s): #функция  
 summa = 0  
 for i in range (len(s)):  
 summa += int (s[i])  
 return summa  
for n in range (1, 100):  
 s = bin(n) [2:] # переводим в двоичную систему  
 s = str(s)  
 s = s + str(f(s) % 2)  
 s = s + str(f(s) % 2)  
 rez = int(s, 2) #переводим в десятичную с.с  
 if rez > 77:  
 print (n)  
 break

**Задание на определение результатов работы простейших алгоритмов.**

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, и 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. Черепахе был дан для исполнения следующий алгоритм: Повтори 4 [Вперёд 9 Направо 90 Вперёд 7 Направо 90].

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

**В задании 6 я представляю два способа решения.**

**1 способ решения через КУМИР**

|  |  |
| --- | --- |
| **использовать** **Черепаха**  **алг**  **нач**  **.** **опустить хвост**  **.** **нц** **4** **раз**  **.** **.** **вперед** (**9**)  **.** **.** **вправо**(**90**)  **.** **.** **вперед**(**7**)  **.** **.** **вправо**(**90**)  **.** **кц**  **кон** |  |

**2 способ более лёгкий через Python**

i = 0 # координаты x, y.  
for x in range (1, 7):  
 for y in range (1, 9):  
 i += 1 #подсчет точек  
print(i)

**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.

**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 других языках с языком 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).

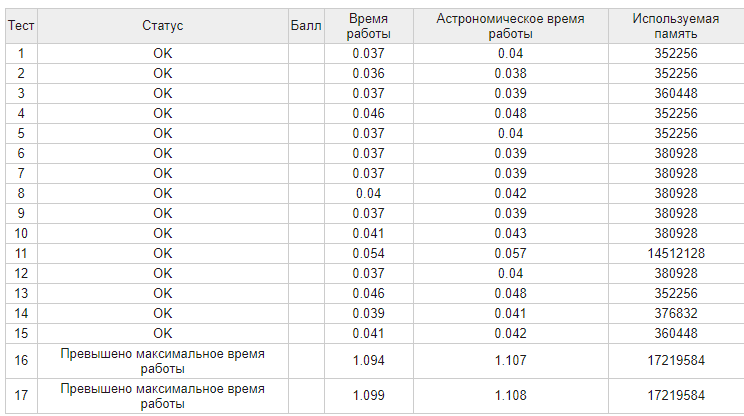


Рис. 1. Результаты прохождения тестов

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

Итак, большее количество задач на экзамене по информатике решается с помощью программирования.

На мой взгляд наиболее лёгким способом решаются задачи с помощью языка программирования Python.

В процессе выполнения работы были решены следующие задачи:

* рассмотрены основы языка программирования Python;
* выделены типы экзаменационных задач по информатике. Задачи решены средствами языка программирования Python;
* проведён сравнительный анализ эффективности других программ с языком программирования Python.

## Список литературы

1. Зыков А.Г., Поляков В.И., Скорубский В.И. Математическая логика. – СПб: НИУ ИТМО, 2013. – 131 с.
2. Zed A. Shaw LEARN PYTHON 3 THE HARD WAY: AVery Simple Introduction to theTerrifying Beautiful World of Computers and Code. 1st edition.
3. Поляков К. Ю., Еремин Е. А. Как нам реорганизовать ЕГЭ по информатике? // Информатика в школе. 2019. № 3.С. 2–7.
4. Сайт «Решу ЕГЭ» [Электронный ресурс].- Режим доступа: https://inf-ege.sdamgia.ru/. – 2.11.2023.