КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ МАТЕМАТИКИ И МЕХАНИКИ ИМ. Н.И. ЛОБАЧЕВСКОГО
Кафедра высшей математики и математического моделирования

А.М. НИГМЕДЗЯНОВА, Е.А. ЕФРЕМОВА

РЕШЕНИЕ ЗАДАЧ ПРОГРАММИРОВАНИЯ ДЛЯ ПОДГОТОВКИ К ЕГЭ ПО ИНФОРМАТИКЕ НА ЯЗЫКЕ PYTHON

Учебно-методическое пособие

Казань 2019

УДК 374.1
УДК 004.43
ББК 74

А.М. Нигмедзянова, Е.А. Ефремова

Решение задач программирования для подготовки к ЕГЭ по информатике на языке Python: учебно-методическое пособие для углубленной подготовки к ЕГЭ по информатике для учащихся и учителей школ, и для студентов педагогического направления.

А.М. Нигмедзянова, Е.А. Ефремова. – Казань: Казан. ун-т, 2019. – 80 с.

Учебно-методическое пособие составлено в соответствии с программой для учителей информатики в старших классах. Пособие содержит основные сведения о языке программирования Python, поурочную разработку проведения элективного курса по информатике в 10-11 классе, типовые примеры из ЕГЭ с решениями на Python.

Оглавление

Введение

Программы должны писаться для людей, которые будут их читать,
а машины, которые будут эти программы исполнять — второстепенны.

Гарольд Абельсон

Первые языки программирования для компьютеров начали разрабатываться с середины 50-х годов XX века. Сейчас же в мире насчитывается более 2500 различных языков программирования, используемых для решения большинства задач.

В последние время в школах на уроках информатики программирование начиналось на языке Pascal. Поскольку программирование и научно-технический прогресс не стоят на месте, на замену языка Pascal пришло много других языков программирования,но почему же, именно Python?

Язык программирования Python достаточно популярен. Это мощный инструмент для создания программ, который доступен даже для начинающего пользователя. Программный код легче для восприятия. Именно поэтому мне кажется, что Python это лучшая замена языку программирования Pascal. Эта методическая разработка позволит учащимся перейти с одного языка программирования на другой, и поможет лучше подготовиться к ЕГЭ.

В первой главе рассмотрены особенности языка программирования Python, его структура.

Во второй главе описывается разработка элективного курса «Решение задач на языке программирования Python» для подготовки к ЕГЭ по информатике.

В третьей главе рассматриваются основные задачи ЕГЭ по информатике, связанные с программированием.

Глава 1

Язык программирования Python.

Язык программирования можно инициализировать как набор команд, направленный на понимание компьютером инструкции к выполнению той или иной программы, написанной на соответствующем языке программирования.

Если же говорить о таком языке программирования как Python, то нельзя не заметить его тенденцию роста популярности. Он используется не только отдельными пользователями, но и целыми компаниями для создания продуктов, приносящих прибыль. Например, компания Google использует Python в своей поисковой системе; платформа YouTube в значительной степени реализована на этом языке. Python ориентирован на повышение производительности разработчика и облегчение задач при написании и читаемости кода. Этот язык программирования был изобретен в 1991 году голландским программистом Гвидо ван Россумом.

1.1 Первые шаги в Python.

Итак, вы установили систему программирования с языком Python. Запустите ее: **Пуск** → **Все программы** → **Python 3.7** → **IDLE (Python 3.7 32-bit)**:

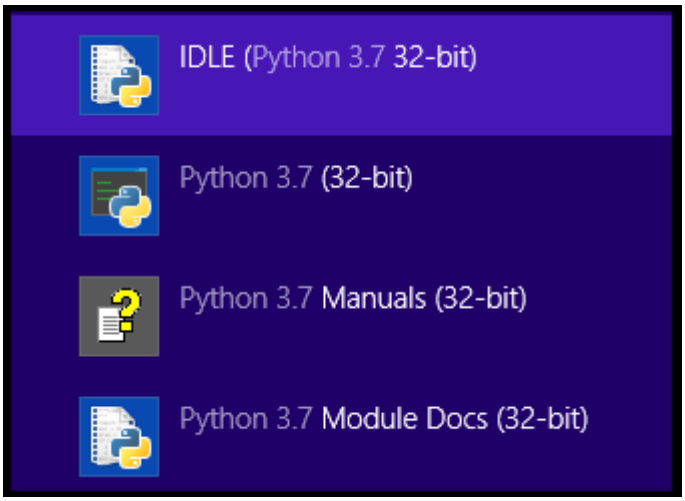


Рис. 1.1

Откроется главное окно **PythonShell** так называемой «интегрированной среды разработки» («IntegratedDeveLopmentEnvironment» – IDLE):

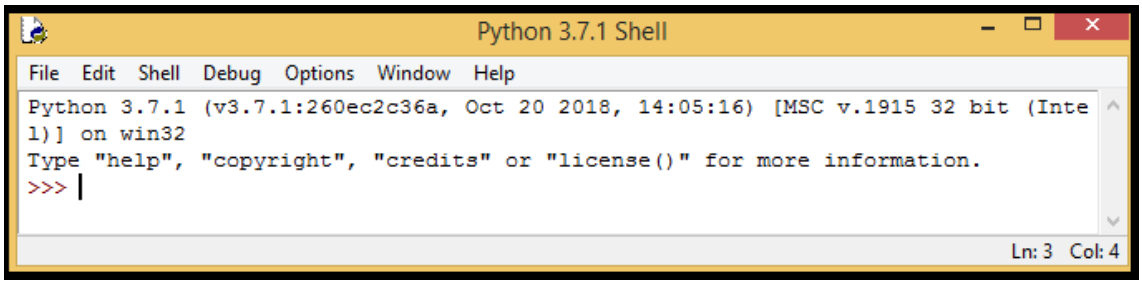


Рис. 1.2 **Окно PythonShell**

Окно **PythonShell** обеспечивает доступ к интерактивному режиму работы, когда каждая введенная команда сразу выполняется и выводится на экран.

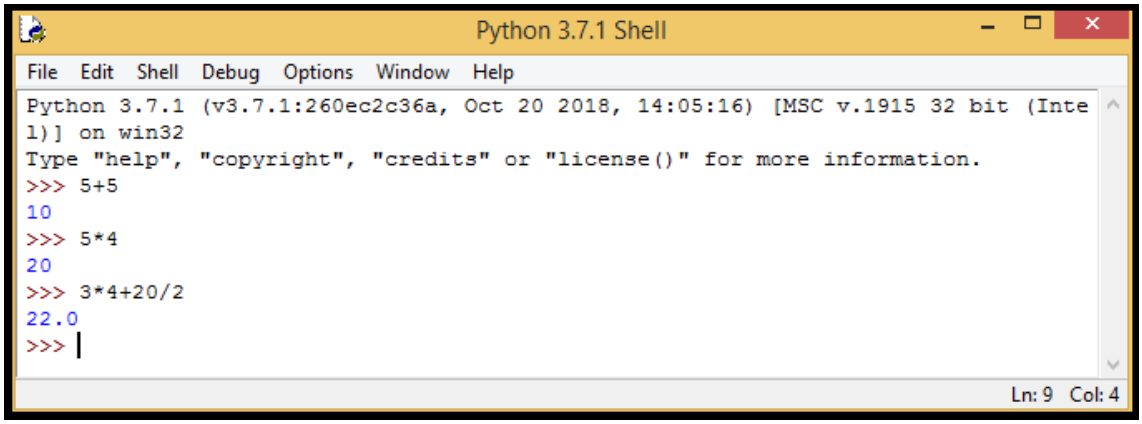


Рис. 1.3

В интерактивном режиме можно писать и выполнять очень простые программы. Но для написания сложных программ используется другой режим работы – программный, когда записывается вся программа и при запуске выполняется целиком (предварительно сохраняя ее в файл на диске (что удобно для повторного выполнения)). Язык программирования Python называют «скриптовым», поэтому саму программу на Python часто называют «скриптом».

Чтобы перейти в программный режим, нужно в меню **File** выбрать пункт **NewFile** или одновременно нажать клавиши <**Ctrl**+**N**>.

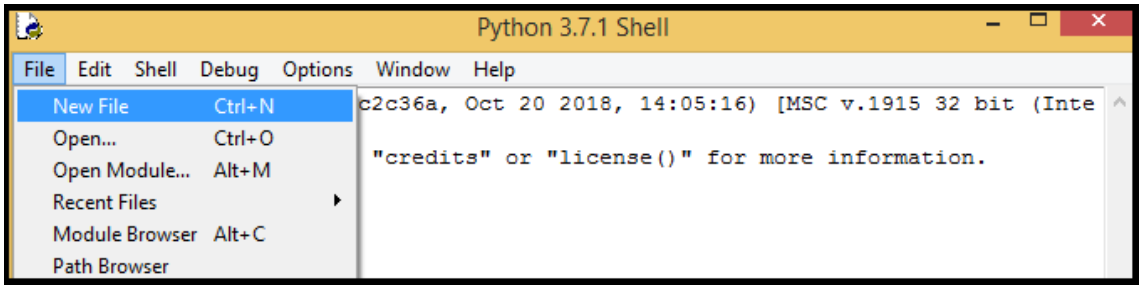


Рис. 1.4

Появится окно для разработки программы (окно редактора):

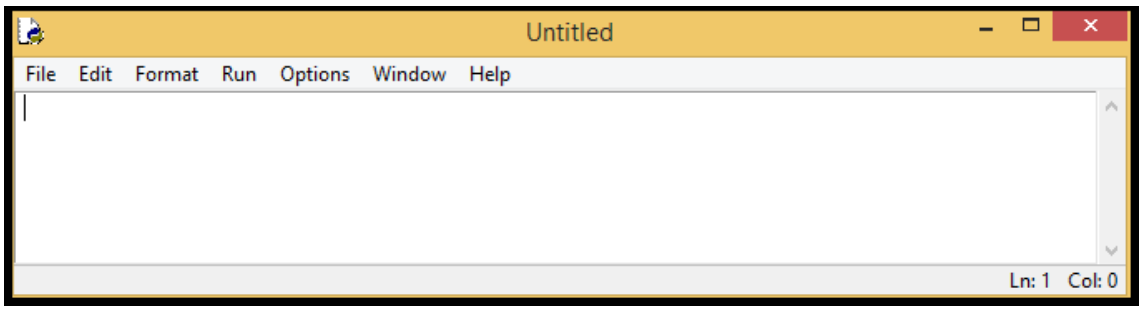


Рис. 1.5

Теперь мы можем написать свою первую программу на Python. Решим стандартную задачу вывода на экран приветствия «Hello, World!».

Программа, как и алгоритм решения задачи, состоит из команд, которые необходимо выполнить для решения задачи. Эти команды в языке Python называются «инструкциями».

Для вывода некоторого текста на экран в языке Python используется конструкция *print()*. Так будет выглядеть текст программы:

```
print (‘Hello, World!’)
```

После начала набора текста программы в окне редактора в заголовке окна **Untitled** будет «окружено» символами “*”:

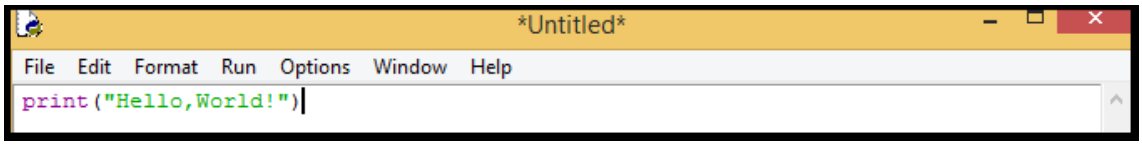


Рис. 1.6

Это говорит о том, что программа в окне еще не записана в файл на диске.

Для сохранения программы нужно в меню **File** выбрать пункт **SaveAs...** или одновременно нажать клавиши <**Ctrl**+**S**>.Появится окно для выбора имени файла с программой и папки, в которой он будет размещен:

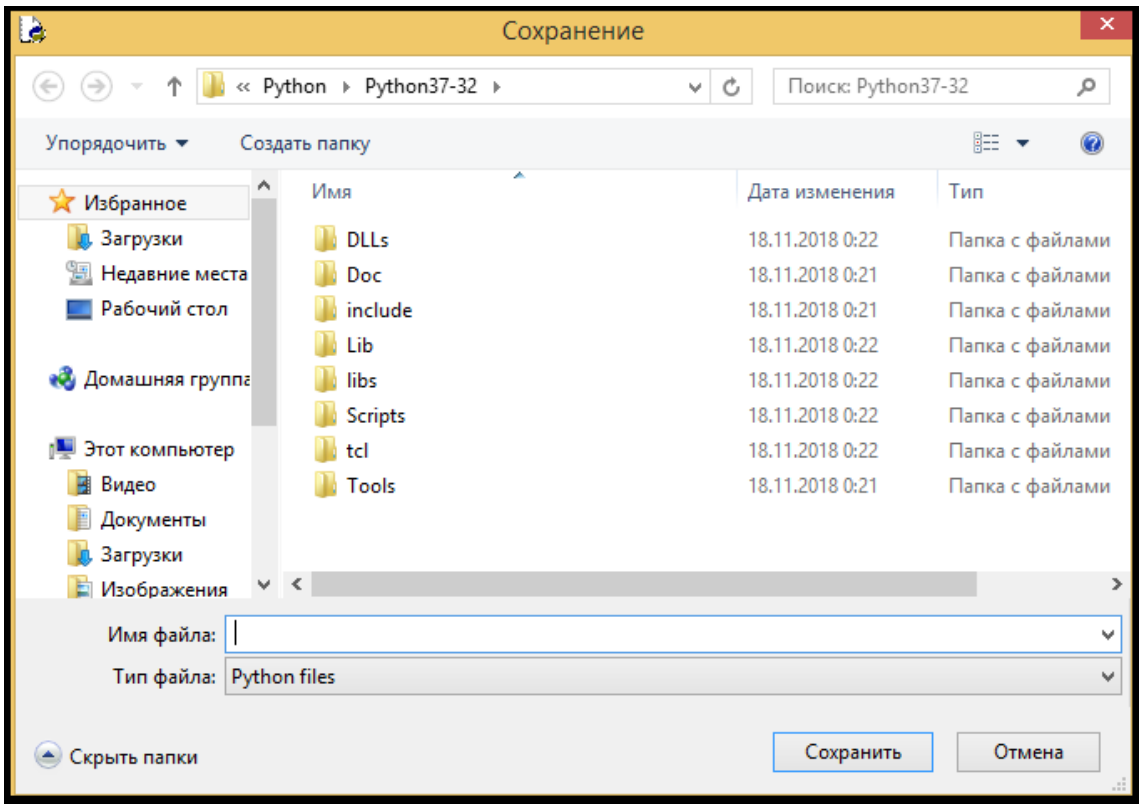


Рис. 1.7

Необходимо придумать «Имя файла», после система добавит к этому имени расширение *.py* и файл с программой разместится с файлами системы программирования.

Чтобы выполнить программу, нужно нажать клавишу <**F5**>. Результат выполнения программы появится в окне **PythonShell**:

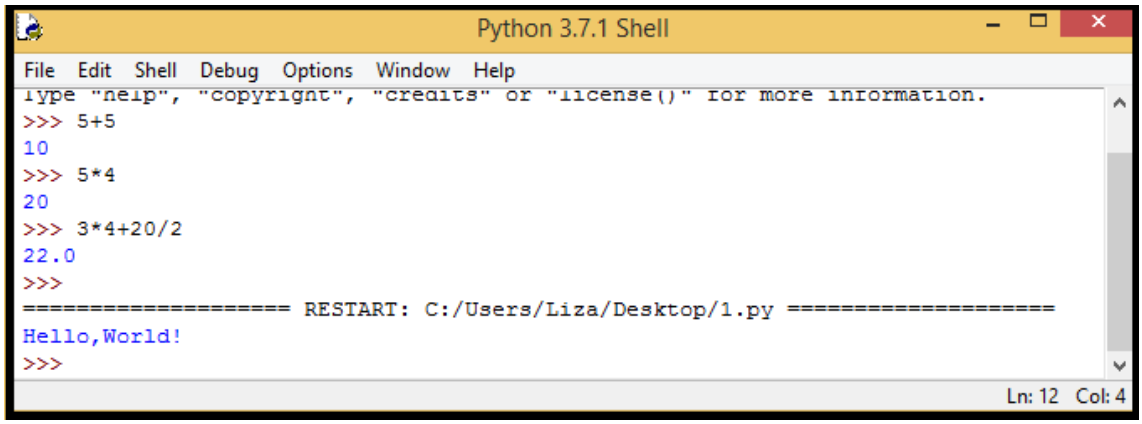


Рис. 1.8

1.2 Вывод и ввод информации на экран.

Для процедуры вывода информации на экран в языке программирования Pyhon используется функция *print()*, а для считывания выведенного пользователем текста используют функцию *input()*.

Давайте напишем программу знакомства на Python. В пустом окне введите следующие три сроки кода:

```
# Твое имя
name=input(‘‘Как тебя зовут?\n’’)
print (‘‘Привет, ’’,name)
```

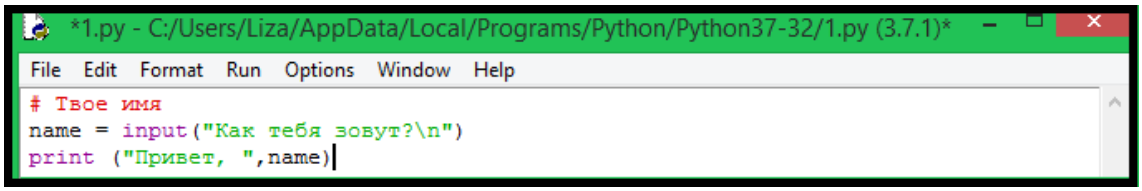


Рис. 1.9

Первая строка называется комментарием и обозначается #. Комментарии для программиста служат заметками или подсказками о действиях в коде. В данной программе комментарий — это просто название программы. Вторая строка просит пользователя ввести свое имя, затем передает это значение в *“name”* и запоминает его. В третьей строке мы просим вывести на экран слово *“Привет, ”* и то, что записано в переменную *“name”*.

После запуска программы перед вами появится следующее:

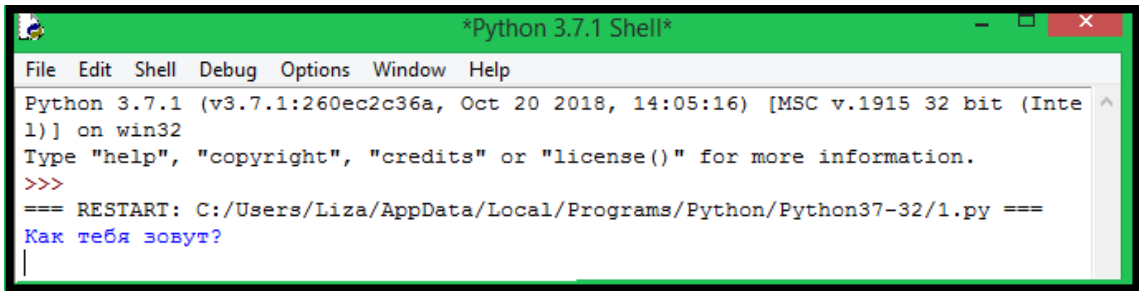


Рис. 1.10

Необходимо ввести имя в строку с курсором:

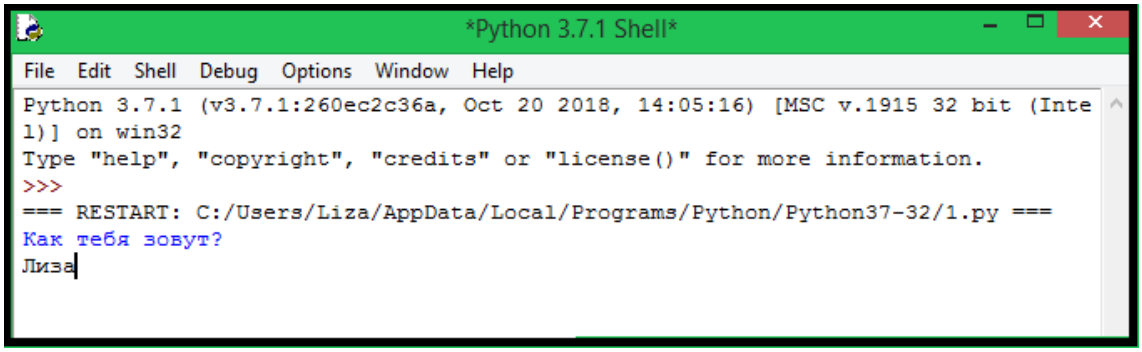


Рис. 1.11

После этого нажав на клавишу **“Enter”** перед вами появится конечный результат написанной программы:

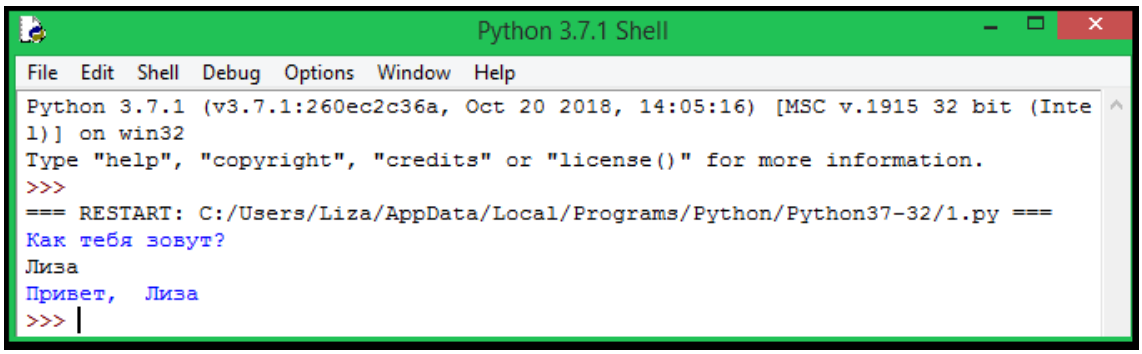


Рис. 1.12

1.3 Переменные величины.

Переменной величиной в Python называют такую величину, которая хранит в себе некоторое переменное значение. Python может запоминать числовые значения (например, 3, 77, 3.14) и строки (буквы, слова, предложения). Ключевым условием для присваивания строк, является заключение этой строки в кавычки (“ ”).

Основные типы данных в Python:

1. boolean – логическое значение (true или false);
2. int – целое число, для хранения которого выделяется 4 байта памяти компьютера;
3. float – число с плавающей точкой, для хранения которого выделяется 8 байт памяти компьютера;
4. str – строки;
5. list – списки;

Операция присваивания в языке Python выполняется с помощью знака равенства “=”. Например, *name = “Иван”* (Рис. 1.13, #1), говорит компьютеру запомнить имя *Иван* и выводить его каждый раз при вызове переменной *name*.

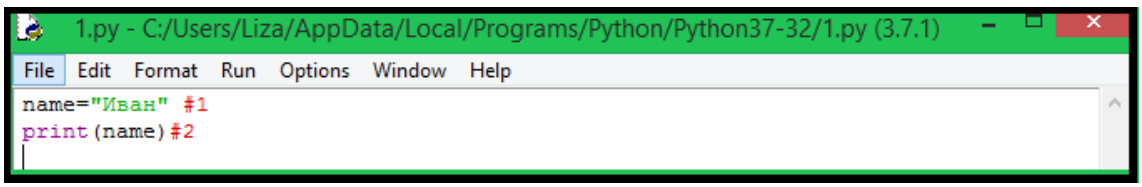


Рис. 1.13

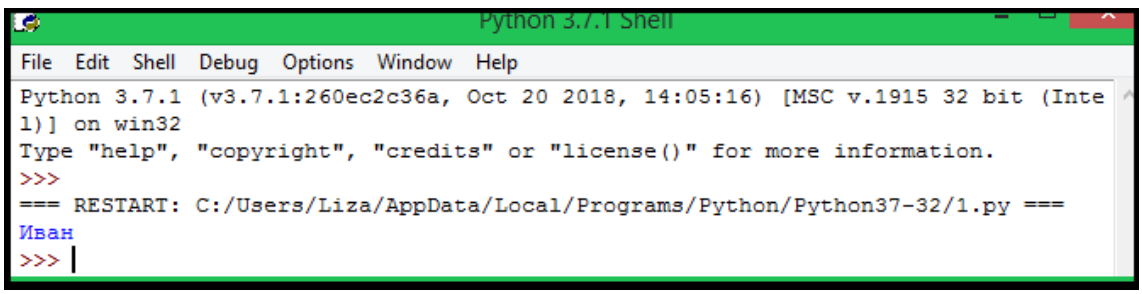


Рис. 1.14

Для того что бы присвоить какое-либо значение в переменную, изначально нужно указать имя этой переменной, знак равенства и только после этого - присваиваемое значение.

При выборе имени для переменной, необходимо придерживаться некоторых правил:

Во-первых, имя переменной всегда должно начинаться с буквы.

Во-вторых, все остальные символы в имени переменной могут быть буквами, цифрами и нижним подчеркиванием “_”. В имени переменной не допускается использование пробелов, это может привести к синтаксической ошибке.

В-третьих, нельзя использовать зарезервированные команды Python для имени переменной. Таких команд не очень много, их легко запомнить: and, as, assert, break, class, continue, def, del, elif, else, except, false, finally, for, from, global, if, import, in, is, lambda, none, nonlocal, not, or, pass, raise, return, true, try, while, with, yield.

В-четвертых, Python очень чувствителен к регистру. Переменные **name**, **Name** и **NAME**, будут представлять собой совершенно разные значения.

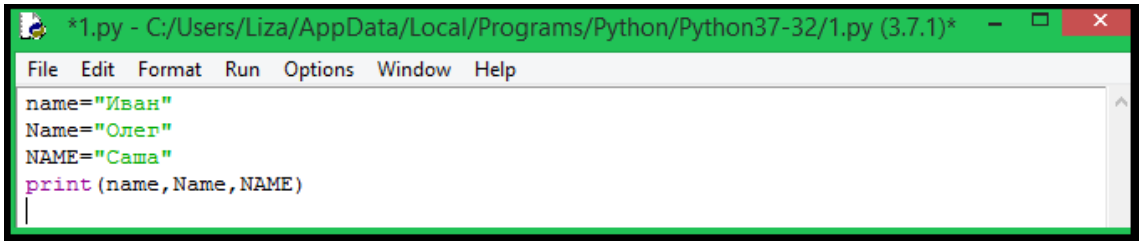


Рис. 1.15

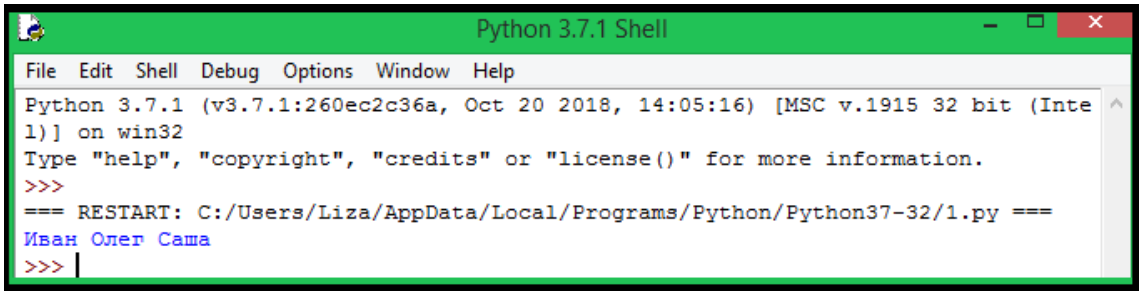


Рис. 1.16

1.4 Условный оператор.

Условная конструкция *if* является основной инструкцией, используемой для выбора среди альтернативных операций на основе результатов проверки.

Проще говоря, условный оператор *if* выбирает, какое действие следует выполнить. Это основной инструмент выбора в языке программирования Python.

Синтаксически изначально записывается часть *if* с условным выражением, далее может следовать одна или несколько необязательных частей *elif (“else if”)* с условными

выражениями и необязательная часть *else*. Условные выражения и часть *else* имеют ассоциативные с ними блоки вложенных инструкций, с отступом относительно основной инструкции. Во время выполнения условного оператора *if* интерпретатор выполняет блок инструкций, ассоциированный с первым условным выражением, только если оно возвращает истину, в противном случае выполняется блок инструкций *else*. Общая форма записи условного оператора выглядит следующим образом:

```
if <test_1>:          #Инструкция if с условным выражением test_1
  <statements_1>      #Ассоциированный блок
elif <test_2>:        #Необязательная часть    elif
  <statements_2>
else:                 #Необязательная часть else:
  <statements_3>
```

При выполнении этой инструкции интерпретатор выполнит вложенные инструкции после той проверки, которая в результате даст истину, или блок *else*, если все проверки дадут ложный результат. На самом деле обе части *elif* и *else* могут быть опущены, и в каждой части может иметься более одной вложенной инструкции.

Условный оператор *if* использует результаты проверки, поэтому рассмотрим некоторые из них:

1. Любое число, не равное нулю, или непустой объект интерпретируется как истина.
2. Числа, равные нулю, пустые и специальный объект *None* интерпретируется как ложь.
3. Операции сравнения и проверки на равенство применяются рекурсивно.
4. Операции сравнения и проверки на равенство возвращают значение *true* или *false*.
5. Логические операторы *and* и *or* возвращают истинный или ложный объект-операнд.

Логические операторы используются для объединения результатов других проверок. В языке программирования Python существует три логических оператора:

```
X and Y     (Истина, если оба значения X и Y истинны)
X or Y      (Истина, если любое из значений X или Y истинно)
not X       (Истина, значение X ложно)
```

Здесь X и Y могут быть любыми значениями истинности или выражениями, которые возвращают значения истинности.

1.5 Циклы.

В данной главе мы встретимся с двумя основными конструкциями организации циклов в языке программирования Python – инструкциями, которые выполняют одну и ту же последовательность действий снова и снова. Первая из них, инструкция *for*, предназначенная для обхода элементов в последовательностях и выполнения блока программного кода для каждого из них; вторая, инструкция *while*, обеспечивает способ организации универсальных циклов.

1.5.1 Цикл for.

Цикл *for* является универсальным итератором последовательностей в языке программирования Python. Он отлично подходит для обхода любых упорядоченных объектов последовательностей. Данный цикл отлично подходит для работы со строками, списками, массивами и другими выстроенными объектами, поддерживающими возможность выполнения итераций.

Цикл *for* начинается со строки заголовка, где указывается переменная для присваивания и объект, обход которого будет выполнен. После следует блок инструкций, которые требуется выполнить.

```
for <target> in <object>:    #Связывает элементы объекта с
                             переменной цикла
  <statements_1>             #Повторяющееся тело цикла (использует
                             переменную цикла)
else:
  <statements_2>
```

Когда интерпретатор выполняет цикл *for*, он поочередно, один за другим, присваивает элементы объекта последовательности переменной цикла и выполняет тело цикла для каждого из них. Для обращения к текущему элементу последовательности в теле цикла используется переменная цикла, перебирающая каждый элемент.

Инструкция *for* также поддерживает необязательную часть *else*, которая выполняется, если выход из цикла производится не инструкцией *break*.

1.5.2 Цикл while.

Цикл *while* является самой универсальной конструкцией для организации итераций в языке программирования Python. Данный цикл продолжает выполнять блок инструкций до тех пор, пока условное выражение продолжает возвращать истину. Конструкция *while* называется циклом, потому что управление циклически возвращается к началу

инструкции, пока условное выражение не вернет ложное значение. Тело цикла продолжает выполняться до тех пор, пока условное выражение истинно, если выражение изначально ложно, то тело цикла не будет выполнено.

Цикл *while* состоит из строки заголовка с условным выражением, тела цикла, содержащего одну или более инструкций с отступами, и необязательной части *else*, которая выполняется, когда не используется инструкция *break*.

В общем виде конструкцию цикла можно представить следующим образом:

```
while <test_1>:          #Условное выражение test_1
  < statements_1>        #Тело цикла
else:                    #Необязательная часть
  <statements_2>         #Выполняется, если условие не выполняется
                         и нет остановки цикла инструкцией break.
```

Инструкция *break* вызывает немедленный выход из цикла. Все что находится в программном коде после этой инструкции, не выполняется. Блок *else* выполняется также в том случае, когда тело цикла ни разу не выполнялось, поскольку в этой ситуации инструкция *break* также не выполняется. В циклах *while* это происходит, когда первая проверка условия в заголовке дает ложное значение.

Приведем пример цикла с инструкцией *break*:

```
while <test_1>:
  <statements_1>
  if <test_2>: break      #Выйти из цикла, пропустив часть else
else:
  <statements_2>          #Выполняется, если не была использована
                          конструкция break.
```

Циклы удобно использовать там, где надо повторно выполнять некоторые действия или многократно обрабатывать данные.

Цикл *for* относится к категории счетных циклов. Обычно он выглядит проще и работает быстрее, чем цикл *while*, поэтому его нужно рассматривать в самую первую очередь, когда возникает необходимость выполнить обход последовательности.

1.6 Функции.

Функции – это самые основные программные структуры в языке программирования Python, которые позволяют многократно использовать программный код с уменьшением его избыточности. Это универсальное средство структурирования программы. Без функций писать эффективные программы практически нереально.

Для описания функции в Python необходимо указать имя функции, описать аргументы функции, описать программный код функции. Описание функции начинается с зарезервированной команды *def*. После указывается название функции, и перечисляются аргументы (после имени функции в круглых скобках):

```
def имя_функции (аргументы):
  команды
```

Тело функции часто содержит инструкцию *return*. Она может располагаться в любом месте тела функции. Данная инструкция завершает работу функции и передает результат вызывающей программе. Команда *return* является необязательной, если ее нет, то работа функции завершается, когда поток управления достигает конца тела функции.

Инструкция *def* в языке программирования Python создает новый объект функции и присваивает этот объект имени. После выполнения этой инструкции появляется возможность вызова функции в программе, добавлением круглых скобок после ее имени. Можно указать один или более аргументов, значение которых будут присвоены именам, указанным в заголовке функции.

Инструкция *def* – это исполняемый программный код, который создает объект функции. При вызове функции, передача объектов производится за счет выполнения операции присваивания, а вычисленные значения возвращаются обратно с помощью инструкции *return*.

Функции представляют собой основной способ избежать избыточности программного кода. Также это основные блоки программного кода многократного использования. Функции позволяют разбить сложную систему на небольшие и легко управляемые части, каждая из которых может разрабатываться отдельно.

Глава 2

Поурочная разработка элективного курса по информатике в 10 - 11 классе.

2.1 Предисловие.

Настоящее пособие имеет целью помочь учителю в планировании и подготовки элективных курсов по информатике в 10-11 классе для подготовки к ЕГЭ. В нем помещены примеры примерного тематического планирования, также содержатся рекомендации по отбору материала на каждый урок.

2.2 Примерное поурочное планирование учебного материала элективных курсов в 10-11 классе при 1 уроке в неделю (33 урока в год)

Для полного понимания структуры языка программирования Python, я рекомендую учителям воспользоваться сайтом Питонтьютор (https://pythontutor.ru).

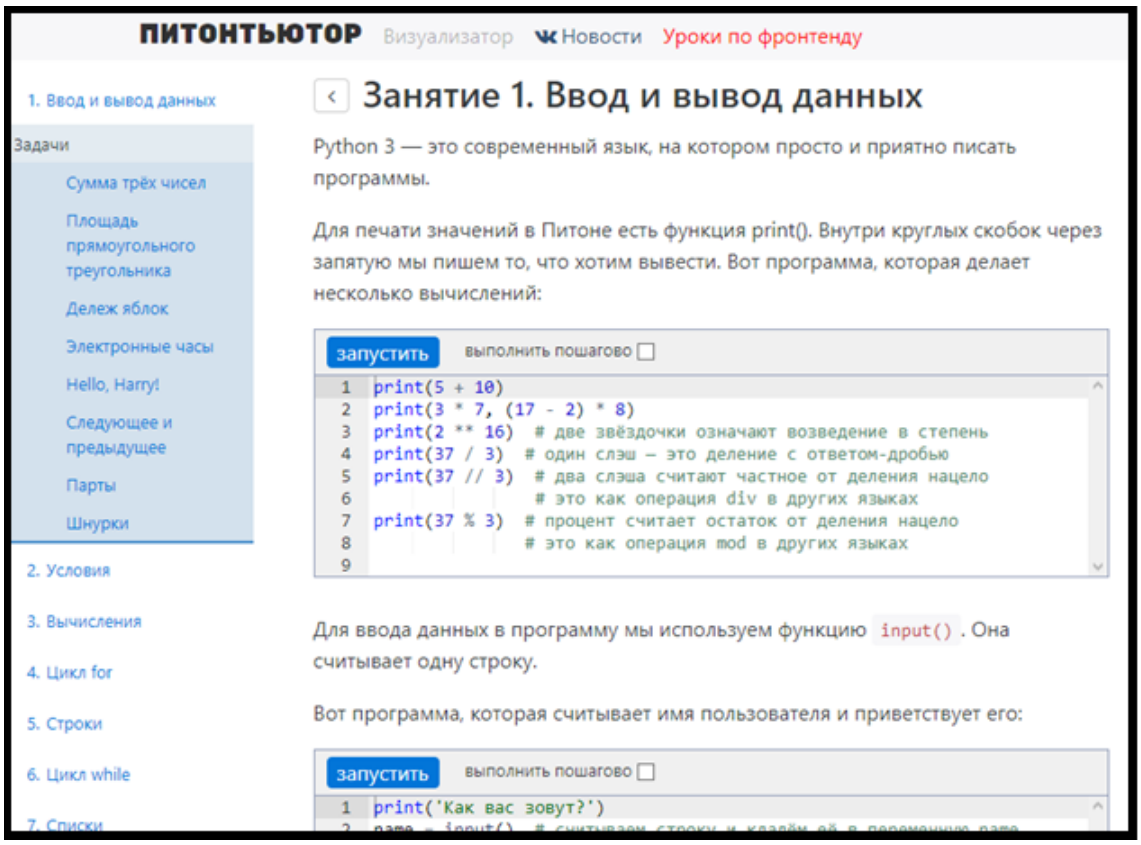


Рис. 2.1

Данный сайт находится в открытом доступе для всех пользователей. Он отлично подходит для первичного изучения и сочетает в себе как теоретический, так и практический материал.

Таблица 2.1

Номер занятия	**Содержание учебного материала**	**Примерные сроки изучения**
1 Четверть. 1 занятие в неделю, 9 занятий за четверть		
Занятие № 1	История языков программирования. Знакомство с Python.	1.09
Сайт Питонтьютор		
Занятие № 2,3	Ввод и вывод данных	8.09-15.09
Занятие № 4,5	Условия	22.09-29.09
Занятие № 6,7	Вычисления	6.10-13.10
Занятие № 8,9	Цикл For	20.10-27.10
Каникулы с 28.10-3.11		
2 Четверть. 1 занятие в неделю, 7 занятий за четверть		
Занятие № 10,11	Строки	10.11-17.10

Занятие № 12,13	Цикл While	24.10-1.12
Занятие № 14-16	Функции и рекурсия	8.12-22.12
Каникулы с 26.12-8.01		
3 Четверть. 1 занятие в неделю, 10 занятий за четверть		
Занятие № 17-19	Двумерные массивы	12.01-26.01
Работа по методической разработке		
Занятие № 20	Задание № 8. Анализ программ	2.02
Занятие № 21	Задание № 11. Рекурсивные алгоритмы	9.02
Занятие № 22	Задание № 19. Обработка массивов и матриц	16.02
Занятие № 23,24	Задание № 20. Анализ программы с циклами и условными операторами	2.03-9.03
Занятие № 25,26	Задание № 21 . Анализ программ с циклами и подпрограммами	16.03-23.03
Каникулы с 25.03-31.03		
4 Четверть. 1 занятие в неделю, 7 занятий за четверть		
Занятие № 27,28	Задание № 24. Исправление ошибок в программе	6.04-13.04
Занятие № 29,30	Задание № 25. Алгоритмы обработки массивов	20.04-27.04
Занятие № 31-33	Решение ЕГЭ	4.05-18.05

Формируемые УУД.

Таблица 2.2

Предметные	**Метапредметные**	**Личностные**
Иметь понятия о предмете информатика, языке программирования Python; знать структуру ввода и вывода данных; основные операции вычислений;	Анализировать и осмысливать изучаемый теоретический материал; уметь извлекать из услышанного на уроке основную информацию; строить .	Овладение системой знаний и умений, необходимых для применения на практике; формирование представления об идеях и методах програм-

разбираться в структуре цикла for и while, условного оператора; иметь представление о структурах строк, функций и двумерного массива; уметь решать задачи ЕГЭ по информатике связанные с программированием.	логические цепочки; оценивать планируемый результат; осуществлять самоконтроль.	мирования; развитие личностных качеств: ясность и точность мысли, критичность мышления, способность к преодолению трудностей; понимание значимости программирования для научно-технического прогресса.

Занятие 1.

Таблица 2.3

Тема	**История языков программирования. Знакомство с Python.**
Этапы урока	
Изучение теоретического материала	1. История возникновения разных языков программирования 2. Языки программирования низкого и высокого уровня 3. Язык программирования Python 4. Структура и особенности языка
Закрепление, решение задач	1. Первые шаги в Python 2. Вывод на экран текста «Hello, World!» 3. Написание программы знакомства на Python
Итог урока	1. Какие языки программирования существуют? 2. Чем отличаются языки программирования низкого и высокого уровня? 3. Что такое Python?

Занятие 2-3.

Таблица 2.4

Тема	**Ввод и вывод данных**
Этапы урока	

Изучение теоретического материала	1. Функция ввода и вывода данных 2. Оператор присваивания 3. Первичное знакомство со строками 4. Типы данных
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 1** 1. Сумма трех чисел 2. Площадь прямоугольного треугольника 3. Дележ яблок **(урок № 3)** 4. Hello, Harry! 5. Следующее и предыдущее 6. Парты
Итог урока	1. Какие функции отвечают за ввод и вывод данных? 2. С какими типами данных мы сегодня познакомились?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 1, задача № 4 (Электронные часы), задача № 8 (Шнурки)

Занятие 4-5.

Таблица 2.5

Тема	**Условия**
	Этапы урока
Изучение теоретического материала	1. Синтаксис условной инструкции 2. Вложенные условные конструкции 3. Операторы сравнения 4. Тип данных bool 5. Логические операторы 6. Каскадные условные инструкции
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 2** 1. Минимум из двух чисел 2. Знак числа

	3. Шахматная доска 4. Високосный год **(урок № 5)** 5. Сколько совпадает чисел 6. Ход ладьи 7. Ход короля 8. Ход слона 9. Ход ферзя 10. Ход коня
Итог урока	1. Как выглядит структура условного оператора? 2. Как записываются операторы сравнения? 3. Что возвращают операторы сравнения?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 2, задача № 5 (Минимум из трех чисел), задача № 12 (Шоколадка), задача № 13 (Яша плавает в бассейне).

Занятие 6-7.

Таблица 2.6

Тема	**Вычисления**
	Этапы урока
Изучение теоретического материала	1. Целочисленная арифметика 2. Действительные числа 3. Библиотека math
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 3** 1. Последняя цифра числа 2. МКАД 3. Дробная часть 4. Первая цифра после точки 5. Конец уроков **(урок № 7)** 6. Стоимость покупки 7. Разность времени 8. Улитка

	9. Число десятков 10. Сумма цифр 11. Часы - 2
Итог урока	1. С помощью, какой операции можно выполнить целочисленное деление? 2. С помощью, какой операции можно извлечь остаток от деления? 3. Что такое библиотека math? 4. Какие операции можно выполнить с ее помощью?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 3, задача № 6 (Автопробег), задача № 15 (Часы - 3), задача № 16 (Проценты).

Занятие 8-9.

Таблица 2.7

Тема	**Цикл For**
	Этапы урока
Изучение теоретического материала	1. Цикл For 2. Функция range 3. Настройка функций print()
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 4** 1. Ряд - 1 2. Ряд – 2 3. Ряд – 3 4. Сумма десяти чисел **(урок № 9)** 5. Сумма кубов 6. Факториал 7. Сумма факториалов 8. Количество нулей
Итог урока	1. Как выглядит структура цикла for? 2. Для чего используется функция range? 3. Для чего нужен параметр sep?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 4,

	задача № 5 (Сумма N чисел), задача № 10 (Лесенка), задача № 11 (Потерянная карточка).

Занятие 10-11.

Таблица 2.8

Тема	**Строки**
Этапы урока	
Изучение теоретического материала	1. Строки 2. Срезы 3. Методы
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 5** 1. Делаем срезы 2. Количество слов 3. Две половинки 4. Переставить два слова 5. Второе вхождение **(урок № 11)** 6. Обращение фрагмента 7. Замена подстроки 8. Удаление символа 9. Замена внутри фрагмента
Итог урока	1. Для чего нужна функция len? 2. К какому типу относится объект после среза? 3. Что возвращает метод find и rfind? 4. Для чего нужен метод replace? 5. Что выводит метод count?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 5, задача № 6 (Удаление фрагмента), задача № 12 (Удалить каждый третий символ)

Занятие 12-13.

Таблица 2.9

Тема	**Цикл While**
Этапы урока	

Изучение теоретического материала	1. Цикл While 2. Инструкции управления циклом 3. Множественное присваивание
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 6** 1. Список квадратов 2. Минимальный делитель 3. Степень двойки 4. Утренняя пробежка 5. Длина последовательности **(урок № 13)** 6. Среднее значение последовательности 7. Максимум последовательности 8. Индекс максимума последовательности 9. Количество четных элементов последовательности
Итог урока	1. Как выглядит синтаксис цикла While? 2. Для чего нужна инструкция break? 3. В чем основная особенность множественного присваивания? 5. Что выводит метод count?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 6, задача № 6 (Сумма последовательности), задача № 12 (Второй максимум).

Занятие 14-16.

Таблица 2.10

Тема	**Функции и рекурсия**
Этапы урока	
Изучение теоретического материала	1. Функции 2. Локальные и глобальные переменные 3. Рекурсия
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 8** 1. Длина отрезка 2. Отрицательная степень

	(урок № 15) 3. Большие буквы 4. Возведение в степень **(урок № 16)** 5. Разворот последовательности
Итог урока	1. Что такое функция? 2. Что возвращают функции max и min? 3. Какие переменные называются глобальными? 4. Что такое рекурсия?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 8, задача № 6 (Числа Фибоначчи).

Занятие 17-19.

Таблица 2.11

Тема	**Двумерные массивы**
Этапы урока	
Изучение теоретического материала	1. Обработка и вывод вложенных массивов 2. Создание вложенных списков 3. Ввод двумерного массива 4. Пример обработки двумерного массива 5. Вложенные генераторы двумерных массивов
Закрепление, решение задач	**Решение задач с сайта Питонтьютор глава 9** 1. Максимум **(урок № 18)** 2. Снежинка 3. Шахматная доска **(урок № 19)** 4. Диагонали, параллельные главной 5. Побочная диагональ
Итог урока	1. Что такое двумерный массив? 2. Как происходит обработка двумерного массива?
Домашнее задание	Решить задачи с сайта Питонтьютор гл. 9, задача № 6 (Поменять столбцы).

Занятие 20.

Таблица 2.12

Тема	**Задание № 8. Анализ программ**
Этапы урока	
Изучение теоретического материала	1. Повторить структуру цикла while
Закрепление, решение задач	**Решение задач из главы 3.1**
Итог урока	1. Выявить основные трудности при решении задач данного типа
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ из блока «Анализ программ».

Занятие 21.

Таблица 2.13

Тема	**Задание № 11. Рекурсивные алгоритмы**
Этапы урока	
Изучение теоретического материала	1. Вспомнить что такое рекурсия 2. Рекурсивные функции
Закрепление, решение задач	**Решение задач из главы 3.2**
Итог урока	1. Выявить основные трудности при решении задач данного типа
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ из блока «Рекурсивные алгоритмы».

Занятие 22.

Таблица 2.14

Тема	**Задание № 19. Обработка массивов и матриц**
Этапы урока	
Изучение теоретического материала	1. Вспомнить что такое двумерные массивы 2. Вспомнить, как происходит замена элементов массива 3. Обсчет массива с накопителем
Закрепление, решение задач	**Решение задач из главы 3.3**

Итог урока	1. Выявить основные трудности при решении задач данного типа
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ из блока «Обработка массивов и матриц».

Занятие 23-24.

Таблица 2.15

Тема	**Задание № 20. Анализ программы с циклами и условными операторами**
Этапы урока	
Изучение теоретического материала	1. Обработка чисел в разных СС 2. Посимвольная обработка десятичных чисел
Закрепление, решение задач	**Решение задач из главы 3.4**
Итог урока	1. Выявить основные трудности при решении задач данного типа
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ из блока « Анализ программы с циклами и условными операторами».

Занятие 25-26.

Таблица 2.16

Тема	**Задание № 21 . Анализ программ с циклами и подпрограммами**
Этапы урока	
Изучение теоретического материала	1. Вспомнить что такое функция 2. Кусочно заданная функция 3. Функция степени 4
Закрепление, решение задач	**Решение задач из главы 3.5**
Итог урока	1. Выявить основные трудности при решении задач данного типа
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ из блока «Анализ программ с циклами и подпрограммами».

Занятие 27-28.

Таблица 2.17

Тема	**Задание № 24. Исправление ошибок в программе**
Этапы урока	
Изучение теоретического материала	1. Последовательность чисел 2. Решение уравнений и неравенств 3. Работа с цифрами числа
Закрепление, решение задач	**Решение задач из главы 3.6**
Итог урока	1. Выявить основные трудности при решении задач данного типа
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ из блока «Исправление ошибок в программе».

Занятие 29-30.

Таблица 2.18

Тема	**Задание № 25. Алгоритмы обработки массивов**
Этапы урока	
Изучение теоретического материала	1. Вспомнить что такое массив 2. Анализ массива с накопителем 3. Поиск максимального и минимального значения
Закрепление, решение задач	**Решение задач из главы 3.7**
Итог урока	1. Выявить основные трудности при решении задач данного типа
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ из блока «Алгоритмы обработки массивов».

Занятие 31-33.

Таблица 2.19

Тема	**Решение ЕГЭ**
Этапы урока	
Закрепление, решение задач	Решение вариантов из ЕГЭ
Итог урока	1. Выявить основные трудности при решении задач ЕГЭ
Домашнее задание	Решить задачи с сайта РЕШУ ЕГЭ.

2.3 Методика ведения элективного курса по теме: «Решение задач ЕГЭ по информатике на языке программирования Python».

Для лучшего усвоения материала курса, предлагается вести занятия по нормативам ФГОС. Это позволит учителю четко регламентировать время каждого этапа, реализовать на занятиях проблемно-поисковый метод, проводить рефлексию учебной деятельности.

Занятие первого типа. Урок открытия новых знаний.

Цели урока: сформировать систему новых понятий, расширить знания учащихся; научить детей новым способам нахождения знаний.

Структура урока

Таблица 2.21

:

1.Организационный этап -Приветствие, подготовка к уроку.	1 мин
2.Актуализация знаний -Проверка домашнего задания, повторение материала прошлых уроков.	5 мин
3.Мотивация к учебной деятельности -Постановка проблемной ситуации, решение которой подводит учащихся к теме урока.	3 мин
4.Объяснение нового материала -Изучение нового материала и способов решения задач.	15 мин
5.Физкультурная минута	2 мин
6.Закрепление изученного материала -Самостоятельное решение учащимися аналогичных задач.	15 мин
7.Рефлексия -Подведение итогов урока	3 мин
8.Домашняя работа -Информация о домашнем задании.	1 мин

Занятие второго типа. Урок рефлексии.

Цели урока: обобщить знания учащихся; систематизировать ЗУН по теме; организовать контроль и самооценку полученных ЗУН.

Структура урока:

Таблица 2.22

:

1.Организационный этап -Приветствие, подготовка к уроку.	1 мин
2.Мотивация к учебной деятельности -Создание условий для возникновения внутренней потребности включения деятельности.	1 мин
3.Актуализация знаний -Повторение ранее изученного материала.	5 мин
4.Самостоятельная работа -Самостоятельное решение задач.	7 мин
5.Физкультурная минута	2 мин
6. Самостоятельная работа -Самостоятельное решение задач.	20 мин
7.Этап целеполагания и построения проекта коррекции выявленных затруднений -Выявление затруднений и поиск их решения	2 мин
8.Рефлексия -Подведение итогов урока.	5 мин
9.Домашняя работа -Информация о домашнем задании.	2 мин

Глава 3

Решение задач ЕГЭ по информатике на языке программирования Python.

В любом варианте ЕГЭ по информатике 27 заданий, из которых 7 связаны с программированием. Именно их я и буду разбирать в этой главе.

3.1 Задание № 8. Анализ программ.

Этот тип заданий представляет собой анализ программ и определение конечного числа, которое будет напечатано в результате выполнения этой программы. Среди всех заданий можно выделить следующие вариации: арифметическая прогрессия; условие выполнения цикла *while*; геометрическая прогрессия. Видимо создатели ЕГЭ надеялись, что ученики будут решать эти задачи с помощью формул алгебраической и геометрической прогрессии. Однако использование данных формул часто приводит к ошибкам и ими удобно пользоваться, когда вы умеете достаточно хорошо решать задачи такого типа напрямую без использования вспомогательных формул. Поэтому для начала решим задачу без использования формул.

Задача № 1. Определите, что будет напечатано в результате работы следующего фрагмента программы:

Python

```
n = 12
s = 5
while n <= 25:
    s += 12
    n += 2
print(s)
```

Рис. 3.1

Цикл *while* выполняется до тех пор, пока истинно условие $n \leq 25$, переменная n

определяет количество итераций цикла. Для начала вручную посчитаем сколько раз выполнится цикл и определим конечное число n.

Таблица 3.1

n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8
12	12+2=14	14+2=16	16+2=18	18+2=20	20+2=22	22+2=24	24+2=26

При n_8 условие цикла не выполняется и соответственно действие $s+ = 12$ так же не будет выполняться.

Теперь найдем значение s. Количество итераций цикла запишем в переменную k, тогда $k = 7$.

Получим $s = k * 12 + 5 = 7 * 12 + 5 = 89$

Рассмотрим еще одну задачу и решим ее двумя способами.

Задача № 2.Определите, что будет напечатано в результате работы следующего фрагмента программы:

Python

```
s = 0
k = 0
while k < 30:
    k += 3
    s += k
print(s)
```

Рис. 3.2

Цикл *while* выполняется до тех пор, пока истинно условие $k < 30$.

Способ № 1. Вручную посчитаем чему будет равен k и s на каждом шаге:

Таблица 3.2

k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}
0	3	6	9	12	15	18	21	24	27	30	33
s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}
0	3	9	18	30	45	63	84	108	135	165	195

При k_{12} условие цикла *while* не выполняется, а s будет равен 165.

Способ № 2.Решим эту же задачу с помощью формулы арифметической прогрессии.

Так как цикл выполняется до тех пор, пока $k < 30$. Воспользуемся методом интервалов и найдем первое натуральное n, при котором нарушается условие:

$$k_n = k_1 + (n - 1)d < 30$$

, где d – разность прогрессии, k_1 - значение k на первом шаге.

$$k_n = 0 + (n-1)3 < 30$$

При $n = 11$ условие цикла *while* не выполняется. Теперь найдем сумму первых n челнов арифметической прогрессии.

$$\mathrm{b} = \frac{2a_1 + (n-1)\,d}{2} n$$

$$\mathrm{s} = \frac{2s_1 + (n-1)\,d}{2} n = \frac{0 + (11-1)\,3}{2} 11 = 165$$

3.2 Задание № 11. Рекурсивные алгоритмы.

Данный тип заданий представляет собой рекурсивные процедуры. Рекурсивные функции – функции, которые в своей записи вызывают сами себя. Иногда в таких задачах дан кусок программного кода, который нужно проанализировать, а иногда только формулы.

Рассмотрим для начала второй случай.

Задача № 1. Алгоритм вычисления значения функции $F(n)$, где n – натуральное число, задан следующими соотношениями:

```
F(1) = 1
F(2) = 2
F(n) = 2 * F(n-1) + (n - 2) * F(n-2), при n >2
Чему равно значение функции F(6)?
```

Рис. 3.3

Данную задачу достаточно легко решить. Нам известно 2 значения это $F(1) = 1$ и $F(2) = 2$, необходимо найти значение функции $F(6)$, при $n > 2$.

$$F(n) = 2 * F(n-1) + (n-2) * F(n-2)(1)$$

Найдем F(6), для которого n=6:

$$F(6) = 2 * F(6-1) + (6-2) * F(6-2)$$

Так как мы не знаем значение $F(5)$ и $F(4)$ возвращаемся к формуле (1) и находим их:

$$F(5) = 2 * F(5-1) + (5-2) * F(5-2)$$

$$F(4) = 2 * F(4-1) + (4-2) * F(4-2)$$

Повторяем эту же процедуру для функции $F(3)$

$$F(3) = 2 * F(3-1) + (3-2) * F(3-2) = 2 * F(2) + 1 * F(1) = 2 * 2 + 1 * 1 = 5$$

Теперь, когда нам известно значение функции $F(3)$, найдем остальные:

$$F(4) = 2 * 5 + 2 * 2 = 14$$

$$F(5) = 2 * 14 + 3 * 5 = 43$$

$$F(6) = 2 * 43 + 4 * 14 = 142$$

Мы нашли искомое значение функции $F(6) = 142$, задача решена.

Теперь рассмотрим задачи, в которых представлен кусок программного кода. Они в свою очередь делятся на 2 типа, в которых есть одна функция и несколько.

Задача № 2.Чему равна сумма напечатанных на экране чисел при выполнении вызова $F(10)$?

Python

```
def F(n):
    if n > 2:
        print(n)
        F(n - 3)
        F(n - 4)
```

Рис. 3.4

Задачи такого типа легче решать с помощью «дерева»:

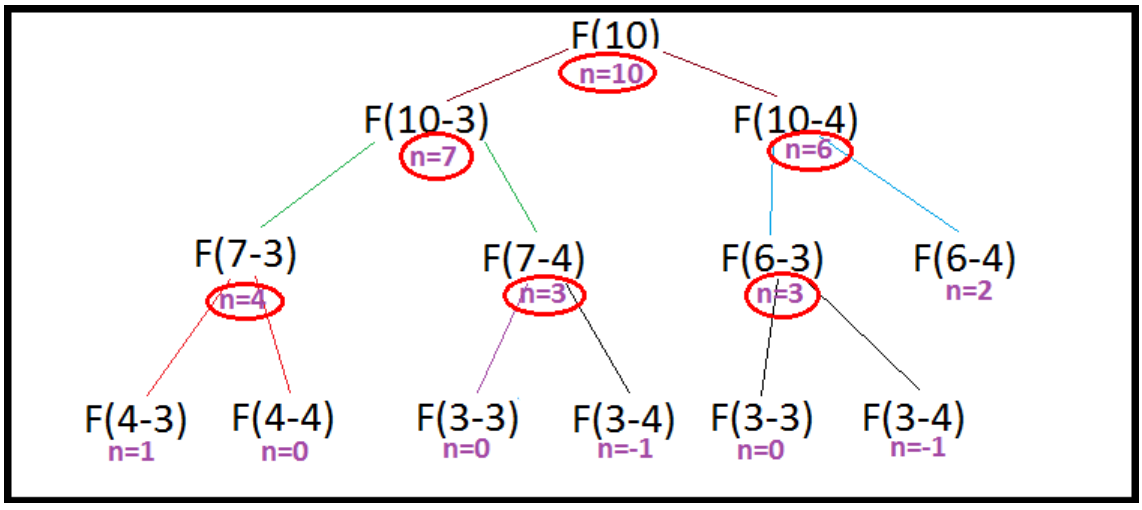


Рис. 3.5

Изначально вызывается процедура $F(10)$, у которой $n = 10$. Так как $10 > 2$, выводится n и вызывается процедура $F(7)$. В данном задании порядок вызова процедур не важен. Далее вызывается процедура $F(6)$. Для $F(7)$ и $F(6)$ условия выполняются, поэтому для процедуры $F(7)$, вызываются $F(4)$ и $F(3)$, а для процедуры $F(6)$, вызываются $F(3)$ и $F(2)$. Для процедур $F(4)$ и $F(3)$ условие выполняется и вызываются процедуры

$F(1)$, $F(0)$ и $F(-1)$. Для этих процедур и процедуры $F(2)$ условия не выполняются и поэтому ветвь на них завершается.

Вывод n находится внутри условия, если $n > 2$, то процедура выводит на экран значение n. Так как нам нужна сумма напечатанных на экране чисел, то необходимо просуммировать все значения n удовлетворяющих нашему условию, а именно $10 + 7 + 6 + 4 + 3 + 3 = 33$. Задача решена.

Задача № 3.Рассмотрим еще одну задачу. Чему будет равно значение, вычисленное при выполнении вызова $F(7)$?

Python

```
def F(n):
  if n > 2:
    return F(n-1)+ G(n-2)
  else: return 1
def G(n):
  if n > 2:
  return G(n-1) + F(n-2)
  else: return 1
```

Рис. 3.6

Для решения будем выписывать формулы, которыми пользуются функции в явном виде. Для функции $F(7)$, $n = 7$, $n > 2$

$$F(7) = F(6) + G(5)$$

Вычислить это значение мы пока не можем, так как не знаем $F(6)$ и $G(5)$. Поэтому найдем значение для $F(6)$, $n = 6$, $n > 2$;

$$F(6) = F(5) + G(4)$$

Продолжим искать значения для $F(5), F(4), F(3)$, для которых $n > 2$:

$$F(5) = F(4) + G(3)$$

$$F(4) = F(3) + G(2)$$

$$F(3) = F(2) + G(1)$$

Для нахождения значения $F(2)$ условие не выполняется, поэтому мы присваиваем ей значение равное одному:

$$F(2) = 1$$

Здесь рекурсия по F заканчивается. Теперь заходим в функцию G и находим значения. Для $G(1)$ условие не выполняется, поэтому так же присваиваем ей значение равное одному, как и для $G(2)$

$$G(1) = 1$$

$$G(2) = 1$$

Теперь можем вычислить значение для $F(3)$ и $F(4)$:

$$F(3) = F(2) + G(1) = 1 + 1 = 2$$

$$F(4) = F(3) + G(2) = 2 + 1 = 3$$

Для того что бы найти значение $F(5)$, необходимо вычислить значение $G(3)$. Так как для функции $G(3)$, условие выполняется, то найдем ее значение по формуле:

$$G(n) = G(n-1) + F(n-2)$$

Тогда

$$G(3) = G(2) + F(1) = 1 + 1 = 2$$

Таким образом, найдем все искомые значения функций:

$$F(5) = F(4) + G(3) = 3 + 2 = 5$$

$$G(4) = G(3) + F(2) = 2 + 1 = 3$$

$$F(6) = F(5) + G(4) = 5 + 3 = 8$$

$$G(5) = G(4) + F(3) = 3 + 2 = 5$$

$$F(7) = F(6) + G(5) = 8 + 5 = 13$$

Значение, полученное при выполнении вызова $F(7) = 13$. Задача решена.

3.3 Задание № 19. Обработка массивов и матриц.

Данный тип заданий характеризуется, как задания повышенного уровня сложности, для их решения необходимо знать следующие темы: цикл *for* со счетчиком и одномерные, двумерные массивы. Можно выделить три вариации структуры задания: алгоритмы, меняющие элементы массива местами; алгоритмы с использованием условного оператора; алгебраические операции с элементами массива, двумерные массивы. Для начала рассмотрим первый тип задач с переменой элементов массива.

Задача № 1. В программе используется одномерный целочисленный массив A с индексами от 0 до 9. Значения элементов равны $1, 2, 5, 8, 9, 3, 4, 0, 7, 6$ соответственно, т. е. $A[0] = 1$, $A[1] = 2$ и т. д.

Определите значение переменной j после выполнения следующего фрагмента программы.

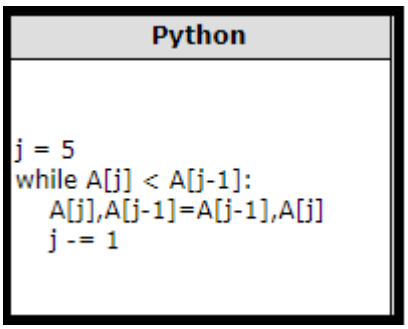


Рис. 3.7

Для начала разберем подробнее, что же такое массив. Массив — это структурный тип данных. Можно сказать, что он состоит из ячеек, у каждой, из которой есть свой индекс. В данной задаче массив будет выглядеть следующим образом

Таблица 3.3

0	1	2	3	4	5	6	7	8	9
1	2	5	8	9	3	4	0	7	6

, где первая строка – индексы массива от 0 до 9, а вторая – элементы массива.

Разберем более подробно программный код. Изначально переменная $j = 5$. Далее начинается выполнение цикла с предусловием. Пока элемент массива A с индексом $[j]$ меньше элемента массива A с индексом $[j-1]$, происходит замена этих элементов местами. Следующая строка $A[j], A[j-1] = A[j-1], A[j]$- это специальная возможность языка программирования Python, позволяющая выполнять эту замену без ввода дополнительного элемента. Последняя строчка, это уменьшение переменной j на единицу при каждом выполнении тела цикла.

Начнем решать данную задачу. При первом повторе цикла $j = 5$. Таким образом, будет выполняться проверка пятого элемента и четвертого. Если $A[5] < A[4]$, то условие цикла выполняется, и они меняются местами.

В нашем случае $A[5] = 3, A[4] = 9, 3 < 9$, следовательно, произойдет замена. $A[5]$ станет равным 9, а $A[4]$ будет равно 3.

Таблица 3.4

0	1	2	3	4	5	6	7	8	9
1	2	5	8	3	9	4	0	7	6

Первый повтор цикла произошел, поэтому $j = 5 - 1 = 4$.

Выполняем проверку дальше, теперь $j = 4$ и мы проверяем третий и четвертый элемент массива. $A[4] = 3, A[3] = 8, 3 < 8$, значит . $A[4] = 8, A[3] = 3, j = 4 - 1 = 3$.

Таблица 3.5

0	1	2	3	4	5	6	7	8	9
1	2	5	3	8	9	4	0	7	6

$A[3] = 3, A[2] = 5, 3 < 5$, значит $A[3] = 5, A[2] = 3, j = 3 - 1 = 2$.

Таблица 3.6

0	1	2	3	4	5	6	7	8	9
1	2	3	5	8	9	4	0	7	6

$A[2] = 3, A[1] = 2, 3 > 2$, условие цикла не выполняется, а значит, замена второго и третьего элемента производиться не будет.

Таблица 3.7

0	1	2	3	4	5	6	7	8	9
1	2	3	5	8	9	4	0	7	6

Переменная $j = 2, A[1] = 2, A[0] = 1, 2 > 1$, условие цикла так же не выполняется.

Таблица 3.8

0	1	2	3	4	5	6	7	8	9
1	2	3	5	8	9	4	0	7	6

В ответе нужно указать значение переменной j после выполнения программы. В нашем случае $j = 2$. Задача решена.

Теперь рассмотрим задачи, связанные с алгебраическими операциями над элементами одномерного и двумерного массива.

Задача № 2. В программе описан одномерный целочисленный массив от 0 до 10.

В начале выполнения этого фрагмента (Рис. 3.8) в массиве находились двухзначные натуральные числа. Какое наибольшее значение может иметь переменная s после выполнения данной программы?

```
s = 0
n = 10
for i in range(0, n-2):
        s = s + A[i] - A[i+3]
```

Рис. 3.8

Нам дан массив с индексами от 0 до 10. Сложность данной задачи в том, что мы не знаем какие именно значения лежали изначально в массиве, но известно, что там были только натуральные двухзначные числа (от 10 до 99). Необходимо узнать какое наибольшее значение может иметь переменная s после выполнения фрагмента программы.

Начнем построчно выполнять код программы. Изначально $s = 0$ и $n = 10$. Далее мы попадаем в цикл for, который будет перебирать все значения i от 0 до $n - 3$ включительно.

Первое значение, которое примет переменная i это 0. Внутри цикла мы выполняем следующее:

$$s = s + A[i] - A[i + 3]$$

При первом выполнении цикла переменная s примет следующее значение:

$$s = 0 + A[0] - A[3] = A[0] - A[3]$$

Теперь $i = 1$ и при следующем выполнении тела цикла переменная s будет равна:

$$s = A[0] - A[3] + A[1] - A[4]$$

При $i = 2$ получим:

$$s = A[0] - A[3] + A[1] - A[4] + A[2] - A[5]$$

Повторяя эту процедуру для $i = 3..7$, переменная s в конце накопит в себе следующее значение:

$$s = A[0] - A[3] + A[1] - A[4] + A[2] - A[5] + A[3] - A[6] + A[4] - A[7] + A[5] - A[8] + A[6] - A[9] + A[7] - A[10]$$

Сейчас в этой большой сумме мы можем сократить некоторые значения, и в итоге переменная s будет равна:

$$s = A[0] + A[1] + A[2] - A[8] - A[9] - A[10]$$

Так как нам нужно узнать какое наибольшее значение может иметь переменная s после выполнения кода программы, то все положительные значения A, а именно $A[0], A[1], A[2]$, будем считать равными самому большому положительному двузначному числу (99), а все отрицательные, а именно $-A[8], -A[9], -A[10]$ – минимальному (10).

Тогда получим конечное максимальное значение, которое может принимать переменная s.

$$s = 99 + 99 + 99 - 10 - 10 - 10 = 267$$

Задача решена. Рассмотрим еще одну задачу с двумя массивами.

Задача № 3. Значения двух массивов $A[1..100]$ и $B[1..100]$ задаются с помощью следующего фрагмента программы:

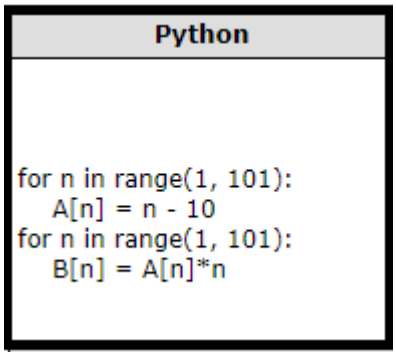


Рис. 3.9

Сколько элементов массива B будут иметь положительные значения?

Решение. Дано два массива A, B с индексами от 1 до 100. Необходимо найти количество положительных значений массива B.

Первый цикл *for*, заполняет элементы массива A следующим образом:

$$A[i] = i * i$$

При $i = 1$, получим:

$$A[1] = 1 * 1 = 1$$

При $i = 2$

$$A[2] = 2 * 2 = 4$$

При $i = 3$

$$A[3] = 3 * 3 = 9$$

При $i = 10$

$$A[10] = 10 * 10 = 100$$

Таким образом, значения элементов массива A, будут равны квадратам соответствующих индексов.

При выполнении второго цикла *for*, будет заполняться массив B

$$B[i] = A[i] - 100$$

При $i = 1$, получим:

$$B[1] = A[1] - 100 = 1 - 100 = -99$$

При $i = 2$

$$B[2] = A[2] - 100 = 4 - 100 = -96$$

При $i = 3$

$$B[3] = A[3] - 100 = 9 - 100 = -91$$

При $i = 10$

$$B[10] = A[10] - 100 = 100 - 100 = 0$$

При $i = 11$

$$B[11] = A[11] - 100 = 121 - 100 = 21$$

Так как нам необходимо найти количество положительных значений массива B, то очевидно, что при значении $i > 10$ все элементы этого массива будут положительными. Следовательно, массив B будет содержать в себе $100-10 = 90$ положительных значений. Это и будет ответом к данной задаче. Задача решена.

Теперь рассмотрим задачу с двумерным массивом.

Задача № 4. Значения элементов двумерного массива A размером $5x5$ задаются с помощью вложенного цикла в представленном фрагменте программы:

Python

```
for i in range(1, 6):
    for j in range(1, 6):
        A[i,j] = i*j
```

Рис. 3.10

Сколько элементов массива будут иметь значения больше 10?

Решение. В задаче необходимо найти количество элементов двумерного массива A, которые будут больше 10. Для начала найдем самое большое значение элемента массива, которое будет меньше 10.

$$A[3, 3] = 3 * 3 = 9; 9 < 10$$

Следовательно, нас интересуют те элементы массива, у которых один индекс ≥ 3, а второй > 3. Этому условию удовлетворяют следующие элементы:

$$A[3, 4] = 3 * 4 = 12; 12 > 10$$

$$A[4, 3] = 4 * 3 = 12; 12 > 10$$

$$A[4, 5] = 4 * 5 = 20; 20 > 10$$

$$A[5, 4] = 5 * 4 = 20; 20 > 10$$

$$A[4, 4] = 4 * 4 = 16; 16 > 10$$

$$A[5,5] = 5 * 5 = 25; 25 > 10$$

$$A[3,5] = 3 * 5 = 15; 15 > 10$$

$$A[5,3] = 5 * 3 = 15; 15 > 10$$

Всего 8 элементов массива будут иметь значение больше 10. Задача решена.

В данном разделе осталось рассмотреть задачи, связанные с использованием условного оператора.

Задача № 5.Значения элементов двумерного массива A[1..100,1..100] задаются с помощью следующего фрагмента программы:

Python

```
for i in range(1, 101):
    for k in range(1, 101):
        if i == k:
            A[i][k] = 1
        else:
            A[i][k] = -1
```

Рис. 3.11

Чему равна сумма элементов массива после выполнения этого фрагмента программы?

Нам дан двумерный массив $A[1..100, 1..100]$, необходимо найти сумму всех элементов массива после выполнения кода программы.

Необходимо понимать, что после заполнения массива у нас получится матрица размерностью 100×100. Массив заполняется с помощью циклов *for* и условного оператора *if*. В условном операторе у нас проверяется условие, если i - тый индекс равен k - ому, то этот элемент равен 1, иначе он равен -1. В нашем случае только элементы, стоящие на главной диагонали будут равны 1. В нашей задаче всего $100 * 100 = 10000$ элементов, из которых только 100 располагается на главной диагонали. Следовательно, после выполнения кода программы сумма элементов массива будет равна:

$$100 * 1 + (10000 - 100) * (-1) = 100 - 9900 = -9800$$

Задача решена.

Задача № 6. Представленный фрагмент программы обрабатывает элементы одномерного целочисленного массива A с индексами от 0 до 9. Перед началом выполнения данного фрагмента эти элементы массива имели значения $2, 4, 8, 6, 1, 7, 2, 3, 2, 7$ (т.е. $A[0] = 2, A[1] = 4, \ldots, A[9] = 7$). Определите значение переменной s после выполнения фрагмента.

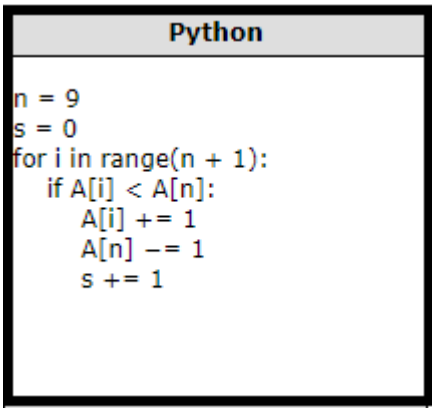


Рис. 3.12

Решение. Дан одномерный целочисленный массив A с индексами от 0 до 9. Нам известны значения элементов данного массива, необходимо определить чему будет равна переменная s после выполнения кода программы. Изначально $s = 0, n = 9$.

В этой задаче массив будет выглядеть следующим образом:

Таблица 3.9

0	1	2	3	4	5	6	7	8	9
2	4	8	6	1	7	2	3	2	7

, где первая строка – индексы массива от 0 до 9, а вторая – элементы массива.

С помощью цикла *for* будет происходить перебор всех элементов массива. Заходя в условный оператор *if* проверяем следующее условие, если

$$A[i] < A[n]$$

, то

$$A[i] += 1$$

$$A[n] -= 1$$

$$s += 1$$

При первом повторе цикла $i = 0$, проверяем условие:

$$[0] < A[9]; 2 < 7$$

Условие выполняется, значит

$$A[0] = 3$$

$$A[9] = 6$$

$$s = 0 + 1 = 1$$

Таблица 3.10

0	1	2	3	4	5	6	7	8	9
3	4	8	6	1	7	2	3	2	6

При $i = 1$

$$A[1] < A[9]; 4 < 6$$

Условие выполняется, следовательно

$$A[1] = 5$$

$$A[9] = 5$$

$$s = 1 + 1 = 2$$

Таблица 3.11

0	1	2	3	4	5	6	7	8	9
3	5	8	6	1	7	2	3	2	5

При $i = 2$

$$A[2] > A[9]; 8 > 5$$

Условие не выполняется.

$$s = 2$$

При $i = 3$

$$A[3] > A[9]; 6 > 5$$

Условие не выполняется.

$$s = 2$$

При $i = 4$

$$A[4] < A[9]; 1 < 5$$

Условие выполняется, следовательно

$$A[4] = 2$$

$$A[9] = 4$$

$$s = 2 + 1 = 3$$

Таблица 3.12

0	1	2	3	4	5	6	7	8	9
3	5	8	6	2	7	2	3	2	4

При $i = 5$

$$A[5] > A[9]; 7 > 4$$

Условие не выполняется.

$$s = 3$$

При $i = 6$

$$A[6] < A[9]; 2 < 4$$

Условие выполняется, следовательно

$$A[6] = 3$$

$$A[9] = 3$$

$$s = 3 + 1 = 4$$

Таблица 3.13

0	1	2	3	4	5	6	7	8	9
3	5	8	6	2	7	3	3	2	3

При $i = 7$

$$A[7] = A[9]; 3 = 3$$

Условие не выполняется.

$$s = 4$$

При $i = 8$

$$A[8] < A[9]; 2 < 3$$

Условие выполняется, следовательно

$$A[8] = 3$$

$$A[9] = 2$$

$$s = 4 + 1 = 5$$

Таблица 3.14

0	1	2	3	4	5	6	7	8	9
3	5	8	6	2	7	3	3	3	2

При $i = 9$

$$A[9] = A[9]; 2 = 2$$

Условие не выполняется.

$$s = 5$$

Таким образом, после выполнения кода программы переменная $s = 5$.

Задача решена.

3.4 Задание № 20. Анализ программы с циклами и условными операторами.

Данный тип заданий связан с алгоритмами, печатающими числа наибольшего либо наименьшего значения вводимого числа. В задачах встречаются две математические операции – целочисленное деление и выведение остатка от деления.

Получение целой части от деления:

$$123//10 = 12$$

Получение остатка от деления:

$$123\%10 = 3$$

Задание № 1. Ниже на пяти языках программирования записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа L и M. Укажите наибольшее из таких чисел x, при вводе которых алгоритм печатает сначала 25, а потом 3.

Python

```
x = int(input())
l=0; m=1
while x > 0:
    l += 1
    if x%2 > 0:
        m *= x%8
    x = x//8
print(m, l)
```

Рис. 3.13

Сложность данной задачи заключается в этих двух строчках:

$$m* = x\%8$$

$$x = x//8$$

А именно в цифре 8.

Для того что бы лучше разобраться в структуре задачи, для начала решим эту задачу с другим условием.

Задача № 1.1.

```
x=int(input())
l=0, m=1
while x>0:
    l+=1
    m*=x%10
    x=x//10
print(m,l)
```

Рис. 3.14

$$m* = x\%10 \ (1)$$

$$x = x//10 \ (2)$$

Строка (1) - это остаток от деления числа x на 10, а (2) - это целочисленное деление числа x на 10.

Допустим, что $x = 846$, тогда

$$x\%10 = 846\%10 = 6$$

$$x//10 = 846//10 = 84$$

А так же пренебрежем следующим условием:

$$if \ x\%2 > 0 \ (3)$$

Решим задачу № 1.1.

Изначально

$$l = 0, m = 1$$

Пусть $x = 846$. Проверяем условие $x > 0$?

$846 > 0$, тогда при первом выполнении тела цикла

$$l = 0 + 1 = 1$$

$$m * x\%10 = 1 * 846\%10 = 1 * 6 = 6$$

$$x = x//10 = 846//10 = 84$$

$$m = 6, x = 84, l = 1$$

Теперь $x = 84, x > 0$. Условие выполняется, значит

$$l = 1 + 1 = 2$$

$$m * x\%10 = 6 * 84\%10 = 6 * 4 = 24$$

$$x = x//10 = 84//10 = 8$$

$$m = 24, x = 8, l = 2$$

$x = 8, x > 0$. Условие выполняется, значит

$$l = 2 + 1 = 3$$

$$m * x\%10 = 24 * 8\%10 = 24 * 8 = 192$$

$$x = x//10 = 8//10 = 0$$

$$m = 192, x = 0, l = 3$$

$x = 0, x < 0$. Условие не выполняется.

На экран выводится $m = 192, l = 3$

Для переменной x значение m - это произведение чисел, а l - количество чисел в нашем числе x.

Задача № 1.2.

Теперь вернем условие (3), тогда программный код будет выглядеть следующим образом:

```
x=int(input())
l=0, m=1
while x>0:
    l+=1
    if x%2>0
        m*=x%10
    x=x//10
print(m,l)
```

Рис. 3.15

В строке (3) проверяется условие нечетности числа.

$$if\ x\%2 > 0\ (3)$$

Если число нечетное, то мы заходим в условный оператор и умножаем переменную m на остаток от деления числа x на 10.

Решим задачу № 1.2.

Изначально

$$l = 0, m = 1$$

Пусть $x = 5432$. Проверяем условие $x > 0$?

$5432 > 0$, тогда при первом выполнении тела цикла

$$l = 0 + 1 = 1$$

$$x\%2 > 0?$$

$$5432\%2 = 0$$

$$0 = 0$$

Число четное, условие не выполняется

$$m = 1$$

$$x = x//10 = 5432//10 = 543$$

$$m = 1, x = 543, l = 1$$

Теперь $x = 543, x > 0$. Условие выполняется, значит

$$l = 1 + 1 = 2$$

$$x\%2 > 0?$$

$$543\%2 > 0$$

$$1 > 0$$

Число нечетное, условие выполняется

$$m * x\%10 = 1 * 543\%10 = 1 * 3 = 3$$

$$x = x//10 = 543//10 = 54$$

$$m = 3, x = 54, l = 2$$

Теперь $x = 54, x > 0$. Условие выполняется, значит

$$l = 2 + 1 = 3$$

$$x\%2 > 0?$$

$$54\%2 = 0$$

$$0 = 0$$

Число четное, условие не выполняется

$$m = 3$$

$$x = x//10 = 54//10 = 5$$

$$m = 3, x = 5, l = 3$$

Теперь $x = 5, x > 0$. Условие выполняется, значит

$$l = 3 + 1 = 4$$

$$x\%2 > 0?$$

$$5\%2 > 0$$

$$1 > 0$$

Число нечетное, условие выполняется

$$m * x\%10 = 3 * 5\%10 = 3 * 5 = 15$$

$$x = x//10 = 5//10 = 0$$

$$m = 15, x = 0, l = 4$$

Теперь $x = 0$, условие не выполняется

На экран выводится $m = 15, l = 4$

Из **Задачи № 1.1.** и **Задачи № 1.2.** можно сделать следующий вывод:

Значение переменной m - это произведение нечетных цифр числа, а значение переменной l - это количество цифр в исходном числе.

Теперь решим еще одну вспомогательную задачу.

Задача № 1.3. Ниже записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа L и M. Укажите наибольшее из таких чисел x, при вводе которых алгоритм печатает сначала 25, а потом 3.

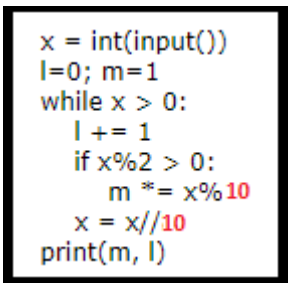


Рис. 3.16

Мы знаем, что m - это произведение нечетных цифр числа, а l - это количество цифр в исходном числе. По условию задачи m должно быть равно 25, а $l = 3$. Следовательно, наше исходное число, должно состоять из 3 цифр, а произведение нечетных цифр должно быть равным 25. Мы можем расписать 25, как произведение 5 и 5. Зачастую, ошибка совершается именно в этом месте, можно предположить, что наше число это 551, но мы можем использовать и четные числа, тогда наибольшим будет число 855. Тогда при вычислении m будут перемножены только нечетные цифры числа (5 и 5), а цифра 8, позволит сделать наше число больше. Для **Задачи № 1.3** ответом будет число 855.

Вернемся к самой первой **Задаче № 1**. Ниже записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа L и M. Укажите наибольшее из таких чисел x, при вводе которых алгоритм печатает сначала 25, а потом 3.

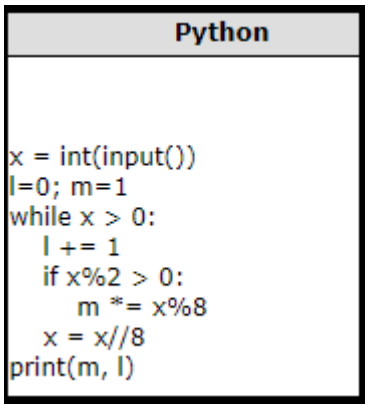


Рис. 3.17

Гораздо легче решить задачу, когда в последних строчках стоит цифра 10. Потому что мы можем составить число из цифр. Но вся трудность заключается именно в остатке при делении на 8.

Почему же именно 10? На самом деле все просто. Цифра 10 это основание десятичной системы счисления. Данный программный код, выполняет те же вычисления, которые мы выполняли с произведением цифр, только делает это в восьмеричной системе счисления.

В восьмеричной системе счисления есть цифры от 0 до 7. Соответственно там есть цифра 5. Но если мы будем использовать число, которое нашли в **Задаче № 1.3**, то

ничего не получится. Так как в восьмеричной системе счисления нет цифры 8, тогда мы можем использовать другое максимально большое четное число из данной системы. Таким числом будет 6, значит наше число в восьмеричной системе счисления это 655. Осталось лишь перевести это число в десятичную систему счисления:

$$655_8 = 6 * 8^2 + 5 * 8^1 + 5 * 8^0 = 384 + 40 + 5 = 429$$

Это и будет наибольшее значение x, удовлетворяющее условию задачи. Задача решена.

Задача № 2. Ниже записан алгоритм. Получив на вход число x, этот алгоритм печатает число M. Известно, что x > 100. Укажите наименьшее такое (т.е. большее 100) число x, при вводе которого алгоритм печатает 26.

Python

```
x = int(input())
L = x
M = 65
if L % 2 == 0:
    M = 52
while L != M:
    if L > M:
        L = L - M
    else:
        M = M - L
print(M)
```

Рис. 3.18

Решение. Нам дан некий алгоритм, в который вводится число, не превышающее 100. Необходимо найти такое наименьшее число, которое после выполнения кода выведет 26.

Цикл завершится, когда переменные L и M будут равны. Значит, мы можем предположить, что перед выводом переменной M, $L = M = 26$.

Изначально $M = 65$, но если мы ввели четное значение x, то $M = 52$. Внутри цикла есть условие, если $L > M$, то $L = L - M$, иначе $M = M - L$.

Рассмотрим последнюю ветвь цикла, в которой после выполнения обе переменные равны 26. Для получения такого результата, необходимо либо от L отнять 26, либо от M. То есть в последней ветви цикла одна переменная должна быть равна 26, а вторая 52. По условию задачи, переменная M может быть равна 52. Очевидно, что будет именно так. Таким образом, от переменной L будет отниматься 52 до тех пор, пока она не станет равной 26. После этого от M будет вычтено 26, и они сравняются.

Так как $x > 100$, то $x = 26 + 52 + 52 = 130$. Проверим это. При первом выполнении тела цикла, переменная $L = 130$. Число 130 - четное значит $M = 52$, $L \neq M . L > M$, а переменная $L = L - M = 130 - 52 = 78$. Теперь $L = 78, M = 52$, $L \neq M$. Выполняем

цикл во второй раз. $L > M$, условие выполняется, следовательно $L = 78 - 52 = 26$. При следующей итерации цикла $L \neq M$, $L < M$, переменной M присвоить следующее значение $M = M - L = 52 - 26 = 26$. После третьего выполнения цикла значение переменных L и M будут равны. Условие входа в тело цикла не выполняется, программа завершается. Таким образом, получим, что минимальное значение x, удовлетворяющее нашему условию это 130. Задача решена. Рассмотрим еще одну задачу на нахождение минимального элемента.

Задача № 3. Ниже записан алгоритм. Получив на вход число x, этот алгоритм печатает два числа: L и M. Укажите наименьшее число x, при вводе которого алгоритм печатает сначала 5, а потом 7.

Python

```
x = int(input())
L = 0
M = 0
while x > 0:
    M = M + 1
    if x % 2 != 0:
     L = L + 1
    x = x // 2
print(L)
print(M)
```

Рис. 3.19

Решение. Так как на выходе мы должны получить два числа 5 и 7, то это означает, что переменная L должна быть равна 5, а переменная M — 7. Нам дан цикл, который зависит от значения x. То есть, пока $x > 0$, тело цикла выполняется. Переменная $M = M + 1$ это счетчик, который при каждом выполнении тела цикла будет увеличиваться на единицу. Следовательно, наш цикл будет выполняться 7 раз. Переменная L увеличивается на 1, если x – нечетное. Это означает, что переменная L считает количество нечетных значений x после целочисленного деления на 2 ($x = x//2$). Цикл остановится, когда переменная $x = 0$. Значит, что при 7 - ой итерации цикла $x = 1$ ($1//2 = 0$). Следовательно, при 6 - том выполнении тела цикла переменная x должна быть равна 2 или 3. Задачи такого типа легче решать с конца.

Построим таблицу возможных значений x на каждом шаге.

Таблица 3.15

Итерация	1	2	3	4	5	6	7
		2					
x	1	либо					
		3					

Рассмотрим все возможные значения, при которых мы можем получить 2 и 3:

$$4//2 = 2$$

$$5//2 = 2$$

$$6//2 = 3$$

$$7//2 = 3$$

Таблица 3.16

Итерация	1	2	3	4	5	6	7
			4				
		2	либо				
			5				
x	1	либо					
			6				
		3	либо				
			7				

Можно заметить, что у нас происходит выполнение двух операций. Это умножение на 2 и умножение на 2 плюс 1. При первой операции мы получаем четные числа, а при выполнении второй – нечетные. Всего в нашем исходном числе должно быть 5 нечетных цифр и только 2 четные. Так как нам нужно найти наименьшее значение, то мы будем использовать верхнюю строку. (Таблица 3.16)

На этом шаге, мы уже использовали максимальное количество четных цифр. Следовательно, все остальные будут нечетными.

$$4 * 2 + 1 = 9$$

Таблица 3.17

1	2	3	4	5	6	7
1	2	4	9			

$$9 * 2 + 1 = 19$$

Таблица 3.18

1	2	3	4	5	6	7
1	2	4	9	19		

$$19 * 2 + 1 = 39$$

Таблица 3.19

1	2	3	4	5	6	7
1	2	4	9	19	39	

$$39 * 2 + 1 = 79$$

Таблица 3.20

1	2	3	4	5	6	7
1	2	4	9	19	39	79

Таким образом, наименьшее значение переменной x, удовлетворяющее условиям задачи – 79. Задача решена.

3.5 Задание № 21. Анализ программ с циклами и подпрограммами.

Этот тип заданий связан с нахождением наибольшего и наименьшего значения функции. Для правильного решения таких задач необходимо понимать, как используются функции в качестве подпрограмм.

Задача № 1. Какое число будет напечатано в результате работы следующей программы?

Python

```
def F(x):
   return 2*(x*x-50)*(x*x-50)+6

a = -11; b = 11
    M = a; R = F(a)
for t in range(a,b+1):
   if F(t) <= R:
       M = t; R = F(t)
print(M+R)
```

Рис. 3.20

В данной задаче нам дана функция F с параметром x. Функция - это подпрограмма, которая возвращает значение, в данном случае функция задана выражением:

$$2 * (x * x - 50) * (x * x - 50) + 6$$

Рассмотрим более подробно сам код программы.

$$a = -11, b = 11$$

$$M = a, R = F(a)$$

Цикл перебирает t от a до b, то есть t изменяется от -11 до 11. Будем считать что t это x, а $F(t)$ это y.

Если

$$F\,(t) \leq R$$

, то

$$M = t$$

$$R = F(t)$$

Проще говоря, переменная R, будет равна значению переменной по оси OY. Для решения этой задачи нам необходимо найти минимальное значение. Рассмотрим график данной функции:

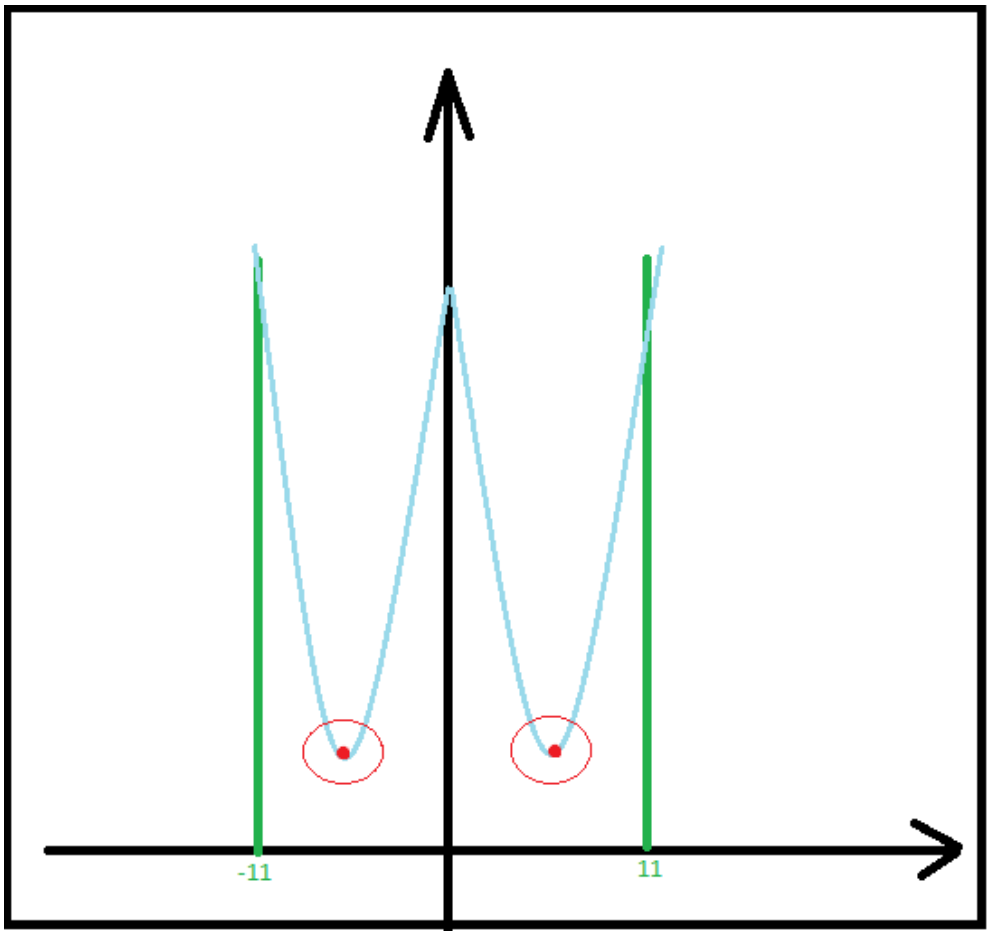


Рис. 3.21

Так как в условии задачи $F(t) \leq R$, то мы будем искать два минимальных значения функции и возьмем только правый.

$$2 * (x * x - 50) * (x * x - 50) + 6$$

Результат данного выражения будет наименьшим при $x = \pm 7$. При любом другом значении число, полученное в скобке, будет больше -1, и тогда полученное число уже точно не будет наименьшим.

Значение переменной $M = 7$, тогда

$$R = F(7) = 2 * (49 - 50) * (49 - 50) + 6 = 8$$

В ответе необходимо записать $M + R = 8 + 7 = 15$.

Задача решена.

Рассмотрим аналогичное задание, которое связанно с поиском максимального значения.

Задача № 2.

Python

```
def f(x):
    return 281 - 2 * (17 + x) * (17 + x)
a = -7
b = 25
M = a
R = f(a)
for t in range(a, b+1):
    if (f(t) > R):
        M = t
        R = f(t);
print(R)
```

Рис. 3.22

При раскрытии скобок, коэффициент при x^2 будет отрицательным. Наибольшее значение будет равно 281, при $x = -17$, но так как эта точка не входит в интервал, то возьмем точку $x = -7$

$$R = F(-7) = 281 - 2 * (17 + x) * (17 + x) = 81$$

Задача решена.

Задача № 3. Напишите в ответе число, равное количеству различных значений входной переменной k, при которых приведённая ниже программа выводит тот же ответ, что и при входном значении $k = 25$. Значение $k = 25$ также включается в подсчёт количества различных значений k.

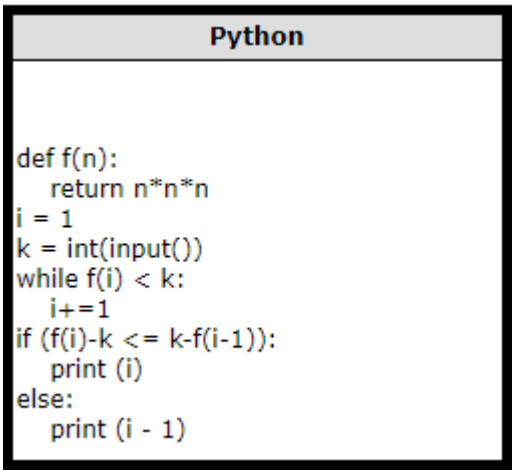


Рис. 3.23

Решение. Для начала определим, что выведет программа при вводе числа 25.

$$f(1) = 1 * 1 * 1 = 1$$

1<25, условие выполняется, $i = 2$, цикл повторяется

$$f(2) = 2 * 2 * 2 = 8$$

8<25, условие выполняется, $i = 3$

$$f(3) = 3 * 3 * 3 = 27$$

27>25, условие не выполняется и цикл завершается.

Теперь проверяем условие:

$$f\,(i) - k \leq k - f\,(i-1)\,(1)$$

$$f\,(3) - 25 \leq 25 - f\,(3-1)$$

$$27 - 25 \leq 25 - 8$$

$$2 \leq 17$$

Условие выполняется, программа выводит значение $i = 3$. Но это не единственное значение. При $i = 4$ условие (1) не выполнится и программа выведет результат $(i-1)$, то есть тоже 3. Необходимо определить какие существуют значения k, при которых $i = 3$ и при которых $i = 4$. Затем выбрать среди них такие, что бы условие выполнялось, и не выполнялось.

$$1 < k, i = 2$$

$$8 < k, i = 3$$

$$27 < k, i = 4$$

$$64 < k, \text{ условие не выполняется}$$

Выпишем все значения k при, которых $i = 3$

$$9, 10 \ldots 26, 27$$

Аналогично для $i = 4$

$$28, 29 \ldots 63, 64$$

Подставим 3 вместо i в условие (1), тогда получим

$$f\,(3) - k \leq k - f\,(2)$$

$$27 - k \leq k - 8$$

$$2k \geq 35$$

$$k \geq 17, 5$$

Для того что бы условие выполнилось для $i = 3$, нужно брать все числа в диапазоне:

$$17, 18 \ldots 26, 27$$

Для $i = 4$ условие выполнится не должно

$$f(4) - k > k - f\ (3)$$

$$64 - k > k - 27$$

$$2k < 91$$

$$k < 45, 5$$

$$28, 29 \ldots 44, 45$$

Осталось только подсчитать количество всех возможных значений для k:

$$27 - 18 + 1 = 10, 45 - 28 + 1 = 18$$

$$10 + 18 = 28$$

Задача решена.

3.6 Задание № 24. Исправление ошибок в программе.

Алгоритмы задания № 20, зачастую повторяются с алгоритмами задания № 24. Если вы умеете достаточно хорошо решать первые, то и со вторыми проблем возникнуть не должно.

Задачи такого типа подразумевают нахождение ошибок в исходном коде программы, а не написание своей собственной. Программа работает и с этими ошибками, но в результате выдает неверный ответ. Во всех задачах допущены 1 или 2 ошибки, и они не синтаксические.

Для начала решим задачу, связанную с последовательностью чисел.

Задача № 1. На обработку поступает последовательность из четырёх неотрицательных целых чисел (некоторые числа могут быть одинаковыми). Нужно написать программу, которая выводит на экран количество делящихся нацело на 4 чисел в исходной последовательности и максимальное делящееся нацело на 4 число. Если делящихся нацело на 4 чисел нет, требуется на экран вывести «NO». Известно, что вводимые числа не превышают 1000. Программист написал программу неправильно. Ниже эта написанная им программа.

Python

```
n = 4
count = 0
maximum = 1000
for i in range(1, n + 1):
    x = int(input())
    if x % 4 == 0:
        count += 1
        if x < maximum:
            maximum = x
if count > 0:
    print(count)
    print(maximum)
else:
    print("NO")
```

Рис. 3.24

Последовательно выполните следующее.

1. Напишите, что выведет эта программа при вводе последовательности: 2 8 4 3.

2. Приведите пример такой последовательности, содержащей хотя бы одно делящееся нацело на 4 число, что при её вводе приведённая программа, несмотря на ошибки, выведет правильный ответ.

3. Найдите допущенные программистом ошибки и исправьте их. Исправление ошибки должно затрагивать только строку, в которой находится ошибка. Для каждой ошибки:

1) выпишите строку, в которой сделана ошибка;

2) укажите, как исправить ошибку, т. е. приведите правильный вариант строки.

Известно, что в тексте программы можно исправить ровно две строки так, чтобы она стала работать правильно.

Достаточно указать ошибки и способ их исправления для одного языка программирования.

Обратите внимание на то, что требуется найти ошибки в имеющейся программе, а не написать свою, возможно, использующую другой алгоритм решения.

Примечание. 0 делится на любое натуральное число.

Решение. В программу вводятся 4 неотрицательных целых числа. Программа должна вывести количество чисел кратных 4 и максимальное число, делящееся нацело на 4. Если таких чисел нет, то программа должна вывести на экран «NO».

Начнем решать эту задачу с конца, а именно найдем ошибки в коде программы. В третьей строке

$$maximum = 1000$$

допущена первая ошибка. Если мы ищем максимум, то мы изначально должны значению максимума присвоить значение минимума. По условию задачи все цифры не отрицательные, следовательно, мы можем присвоить значению $maximum$ минимальное число равное 0 или -1. Нужно запомнить, что если мы ищем максимум, то этому значению мы должны всегда присваивать минимальное значение и наоборот.

Вторая строка

$$count = 0$$

это количество. То есть в нашей задаче это количество чисел кратных 4. Изначально эта строка должна быть равна 0. Здесь ошибок нет.

Посмотрим на последний условный оператор:

$$if\ count > 0$$

Если при выполнении кода программы найдется хоть одно число кратное 4, то на экран выведется количество таких цифр и максимальное число. В противном случае, программа должна вывести на экран «NO». Верно, следовательно, в последнем куске так же нет ошибки.

Проверим правильность работы цикла for.

$$for\ i\ in\ range(1, n + 1)\ (1)$$

$$x = int(input())\ (2)$$

$$if\ x\%4 == 0\ (3)$$

$$count+ = 1\ (4)$$

$$if\ x < maximum : (5)$$

$$maximum = x\ (6)$$

В (1) и (2) строке ошибок нет, проверяем (3) и (4). Если x кратно 4, то мы прибавляем единицу в количество. Действительно, мы ищем числа кратные 4, если такие есть, то наш счетчик должен увеличиться на 1.

Проверяем (5) и (6) строку. Если x меньше максимума, то значению максимума присваивается x. Здесь и скрывается вторая ошибка.

Как осуществляется поиск максимума и минимума? Если что-то больше максимума, то значению максимума присваивается какое-то значение. Аналогичная ситуация с поиском минимального значения. Если что-то меньше минимума, то значению минимума присваивается какое-то значение.

Следовательно, в коде программы находится не максимальное значение, а минимальное число кратное 4. Для исправления этой ошибки, нужно в (5) строке изменить знак на противоположный. Таким образом, данная программа находит минимальное кратное 4 число и выводит количество цифр кратных 4.

Вернемся к условиям задачи и выполним первый пункт. Найдем, что именно выведет программа при вводе последовательности 2 8 4 3.

Последовательно выполните следующее.
1. Напишите, что выведет эта программа при вводе последовательности: 2 8 4 3.
2. Приведите пример такой последовательности, содержащей хотя бы одно делящееся нацело на 4 число, что при её вводе приведённая программа, несмотря на ошибки, выведет правильный ответ.
3. Найдите допущенные программистом ошибки и исправьте их. Исправление ошибки должно затрагивать только строку, в которой находится ошибка. Для каждой ошибки:
1) выпишите строку, в которой сделана ошибка;
2) укажите, как исправить ошибку, т. е. приведите правильный вариант строки.

Рис. 3.25

1) Так как количество кратных чисел считается верно, то $count = 2$. Программа ищет минимальное значение кратное 4, тогда $maximum = 4$.

Ответ: 1) 2 4

2) Необходимо привести пример такой последовательности, в которой есть число кратное 4, и при вводе которой программа действительно выведет максимальное число кратное 4. Это возможно, если число кратное 4 единственное. Можно привести следующие примеры последовательностей 1 2 3 4, 1 1 1 4, 2 3 1 4, 4 4 1 1. Требуется найти хотя бы одну такую последовательность.

Ответ: 1) 2 4; 2)1 2 3 4

3) Ошибки в коде программы мы нашли ранее, осталось их записать.

Ответ: 1) 2 4; 2)1 2 3 4; 3) $if\ x < maximum :=> if\ x > maximum :; maximum = 1000 => maximum = -1$.

Задача решена.

Рассмотрим пару задач, связанных с работой над цифрами числа.

Задача № 2. Требовалось написать программу, при выполнении которой с клавиатуры считывается натуральное число N, не превосходящее 10^9, и выводится сумма цифр этого числа. Программист торопился и написал программу неправильно.

Python

```
n = int(input())
sum = 0
while n >= 9:
   digit = n % 10
   sum += digit
   n //=10;
print(sum)
```

Рис. 3.26

Последовательно выполните следующее.

1. Напишите, что выведет эта программа при вводе числа 352.

2. Найдите все ошибки в этой программе (их может быть одна или несколько). Укажите все строки (одну или более), содержащие ошибки, и для каждой такой строки приведите правильный вариант. Обратите внимание, что требуется найти ошибки в имеющейся программе, а не написать свою, возможно, использующую другой алгоритм решения. Исправление ошибки должно затрагивать только строку, в которой находится ошибка.

3. Укажите одно число для которого эта программа будет работать верно.

Решение. С клавиатуры вводится некоторое число, после выполнения кода программы на экране должна появиться сумма цифр вводимого числа. В исходном коде переменная *digit* хранит в себе остаток от деления, а в переменной sum происходит суммирование всех остатков полученных при выполнении цикла *while*. Программа достаточно простая, для правильного решения, необходимо помнить математические операции // и %.

Начнем решение задачи с первого пункта.

1) Найдем, что выведет эта программа при вводе числа 352.

$$n = 352, sum = 0$$

Проверяем условие для входа в цикл *while*:

$$352 > 9$$

Условие выполняется, тогда:

$$digit = n\%10$$

$$digit = 2$$

$$sum+ = digit$$

$$sum = 0 + 2 = 2$$

$$n// = 10$$

$$n = 35$$

Теперь

$$n = 35, sum = 2$$

$$35 > 9$$

Условие выполняется, тогда:

$$digit = n\%10$$

$$digit = 5$$

$$sum+ = digit$$

$$sum = 2 + 5 = 7$$

$$n// = 10$$

$$n = 3$$

После второго выполнения тела цикла:

$$n = 3, sum = 7$$

Условие для третьего входа в цикл не выполняется:

$$3 < 9$$

Выводится значение $sum = 7$

Ответ: 1) 7; 2) При выполнении кода программы мы поняли, что в переменной sum копится сумма всех цифр числа кроме старшего. Для вывода суммы всех цифр числа, необходимо изменить 3 строку.

$$while\ n \geq 9 :=> while\ \ n > 0 :$$

Ответ: 1)7; 2) $while\ n \geq 9 :=> while\ \ n > 0$:;

3) Данная программа будет работать верно, для всех чисел, начинающихся с 9 (913, 999, 9, 90, ...). В ответ можно указать любое такое число не превосходящее 10^9.

Ответ: 1)7; 2)$while\ n \geq 9 :=> while\ \ n > 0$:; 3) 901.

Задача решена.

Задача № 3. Требовалось написать программу, при выполнении которой с клавиатуры считывается натуральное число N, не превосходящее 10^9, и выводится сумма чётных цифр в десятичной записи этого числа или 0, если чётных цифр в записи нет. Ученик написал такую программу:

Python

```
n = int(input())
s = 0
while n > 1:
    if N % 2 == 0:
        s = N % 10
    N //= 10
print(s)
```

Рис. 3.27

Последовательно выполните следующее.

1. Напишите, что выведет эта программа при вводе числа 1984.

2.Приведите пример числа, при вводе которого программа выдаст верный ответ.

3. Найдите в программе все ошибки (их может быть одна или несколько).

Решение. Решим эту задачу с первого пункта.

Определим, что именно выведет программа при вводе числа 1984.

$$N = 1984, s = 0$$

Проверяем условие для входа в цикл $while$:

$$1984 > 1$$

Условие выполняется, далее происходит проверка на четность:

$$if\ N\%2 == 0$$

Число четное

$$s = N\%10$$

$$s = 4$$

$$N// = 10$$

$$N = 198$$

При второй итерации цикла, получим:

$$s = 8$$

$$N = 19$$

При третьем выполнении, условие входа в цикл *while* выполняется:

$$19 > 1$$

А условие входа в условный оператор нет, так как число нечетное.

$$N = 1$$

При четвертой итерации оба условия не выполняются, поэтому мы выходим из цикла и выводим значение переменной $s = 8$.

Ответ: 1) 8

2) Нужно привести пример такого числа, при котором программа выдаст верный ответ. Правильный ответ можно получить в 4 случаях:

А) Если в числе нет чётных цифр (1357)

Б) Если в числе чётная цифра только одна (1349)

В) Если все чётные цифры в числе равные 0 (1050)

Г) Если в числе есть нули и только одна цифра > 0. И если все нули расположены правее ненулевой чётной цифры. (1670)

Следовательно, ответом к этому пункту может быть любое число, удовлетворяющее хотя бы одному из 4 случаев.

Ответ: 1) 8; 2) 1200

3) Данная программа выводит первую четную цифру в записи числа. Для вывода суммы всех чётных цифр, необходимо изменить 5 строку кода:

$$s = N\%10 => s+ = N\%10$$

Ответ: 1) 8; 2) 1200; $s = N\%10 => s+ = N\%10$.

Задача решена.

Так же рассмотрим пару задач, связанных с решением уравнений и неравенств.

Задача № 4. Для заданного положительного вещественного числа A необходимо найти максимальное целое число K, при котором выполняется неравенство (Рис. 3.28)

$1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{K}<A.$
(при K = 0 сумма считается равной 0).
Для решения этой задачи ученик написал такую программу.

Python

```
a = float(input())
k = 1
s = 1
while s < a:
   s = s + 1.0/k
   k = k + 1
print(k)
```

Рис. 3.28

Последовательно выполните следующее.

1. Напишите, что выведет эта программа при вводе числа 1.2.

2. Приведите пример числа, при вводе которого программа даст верный ответ.

3. Найдите в программе все ошибки (их может быть одна или несколько).

Решение. Эту задачу будем решать так же с первого пункта.

1) Необходимо определить, что выведет программа при вводе числа 1.2.

$$a = 1.2$$

$$k = 1, s = 1$$

$$while\ s < a:$$

$$1 < 1.2$$

Условие выполняется, тогда получим

$$s = s + \frac{1}{k}$$

$$s = 1 + \frac{1}{1} = 2$$

$$k = k + 1$$

$$k = 1 + 1 = 2$$

Снова проверяем условие, оно не выполняется. Выводим значение переменной $k = 2$.

Ответ: 1) 2

2) При $a = 1.2$ данное неравенство не выполняется:

$$1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k} < a$$

$$1 + \frac{1}{2} < 1.2$$

$$1.5 > 1.2$$

Программа даст верный ответ при $a = 1.6$

Ответ: 1) 2; 2) 1,6

3) В данном программном коде 2 ошибки:

$$s = 1 => s = 0$$

Так как переменная s оказалась на единицу больше вводимой переменной a, необходимо переменную s обнулить.

$$while\ (k) => while\ (k - 2)$$

При исходном выводе программа находит минимальною значение k, при котором неравенство не выполняется. А нам необходимо найти максимальное значение, при котором неравенство выполняется. Здесь происходит увеличение переменной k на единицу. Так как изначально происходит увеличение переменной s и только после этого происходит увеличение переменной k, то это так же приводит к увеличению переменной k на единицу. Именно поэтому необходимо вычесть из переменной k перед выводом 2 единицы.

Ответ: 1) 2; 2) 1,6; 3) $s = 1 => s = 0$; $while\ (k) => while\ (k - 2)$.

Задача решена.

Задача № 5. Требовалось написать программу, которая решает уравнение $x^2+c = 0$ относительно x для любого числа c, введенного с клавиатуры. Все числа считаются действительными. Программист торопился и написал программу неправильно.

Python

```
c = float(input())
x = float(input())
if c > 0:
    print("нет решений")
else:
    print('x=',sqrt(-c),
' или x=',-sqrt(-c))
```

Рис. 3.29

Последовательно выполните три задания:

1) Приведите пример таких чисел c, x, при которых программа неверно решает поставленную задачу.

2) Укажите, какая часть программы является лишней.

3) Укажите, как нужно доработать программу, чтобы не было случаев ее некорректной работы. (Это можно сделать несколькими способами, поэтому можно указать любой способ доработки исходной программы).

Решение.

1) Если переменные x и c будут равны 0, то программа будет неверно решать поставленную задачу.

2) Так как по условию задачи необходимо ввести с клавиатуры только значение переменной c, то вторая строка будет лишней.

3) Программу можно доработать, добавив в нее еще одну проверку.

$$if\ c > 0:$$

$$print(\text{"нет решения"})$$

$$elif\ c = 0:$$

$$print(\text{"}x = 0\text{"})$$

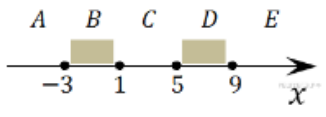
$$else$$

$$print('x =', sqrt(-c), ' x =', sqrt(-c))$$

Задача решена.

Рассмотрим последнюю задачу в данном блоке, связанную с числовой прямой.

Задача № 6. Требовалось написать программу, при выполнении которой с клавиатуры считывается координата точки на прямой (x — действительное число) и определяется принадлежность этой точки одному из выделенных отрезков B и D (включая границы). Программист торопился и написал программу неправильно.



Python

```
x = int(input())
if x >= -3:
    if x <= 9:
        if x > 1:
            print("не принадлежит")
        else:
            print("принадлежит")
```

Область	Условие 1 (x >= −3)	Условие 2 (x <= 9)	Условие 3 (x > 1)	Программа выведет	Область обрабатывается верно
A					
B					
C					
D					
E					

Рис. 3.30

Последовательно выполните следующее.

1. Перерисуйте и заполните таблицу, которая показывает, как работает программа при аргументах, принадлежащих различным областям (A, B, C, D и E). Границы (точки –3, 1, 5 и 9) принадлежат заштрихованным областям (B и D соответственно).

В столбцах условий укажите «Да», если условие выполнится; «Нет», если условие не выполнится; «—» (прочерк), если условие не будет проверяться; «не изв.», если программа ведет себя по-разному для разных значений, принадлежащих данной области. В столбце «Программа выведет» укажите, что программа выведет на экран. Если программа ничего не выводит, напишите «—» (прочерк). Если для разных значений, принадлежащих области, будут выведены разные тексты, напишите «не изв.». В последнем столбце укажите «Да» или «Нет».

2. Укажите, как нужно доработать программу, чтобы не было случаев её неправильной работы. (Это можно сделать несколькими способами, достаточно указать любой способ доработки исходной программы.)

Решение. Для решения задачи, необходимо перерисовать данную таблицу и заполнить ее правильно. Исправить ошибку в коде программы для корректной работы всего кода.

Определимся с возможными значениями, которые мы будем проверять в разных областях.

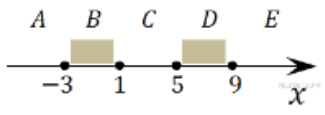
$$A \ni (-\infty; -3)$$

$$B \ni [-3; 1]$$

$$C \ni (1; 5)$$

$$D \ni [5; 9]$$

$$E \ni (9; +\infty)$$



Python

```
x = int(input())
if x >= -3:
    if x <= 9:
        if x > 1:
            print("не принадлежит")
        else:
            print("принадлежит")
```

Область	Условие 1 (x >= –3)	Условие 2 (x <= 9)	Условие 3 (x > 1)	Программа выведет	Область обрабатывается верно
A					
B					
C					
D					
E					

Рис. 3.30

Таблица 3.21

Область	Условие 1 ($x \geq -3$)	Условие 2 ($x \leq 9$)	Условие 3 ($x > 1$)	Программа выдает	Область обрабатывается верно
A					
B					
C					
D					
E					

Сделаем подробный разбор для заполнения первых двух областей, остальные заполняются аналогично.

Возьмем значение переменной x из определенной ранее области $A \ni (-\infty; -3)$

Пусть $x = -5$, тогда выполним код программы:

Python

```
x = int(input())
if x >= -3:
    if x <= 9:
        if x > 1:
            print("не принадлежит")
        else:
            print("принадлежит")
```

Рис. 3.31

Проверяем условие для входа в первый условный оператор if:

$$-5 \geq -3$$

Условие не выполняется, таблица примет следующий вид:

Таблица 3.22

Область	Условие 1 ($x \geq -3$)	Условие 2 ($x \leq 9$)	Условие 3 ($x > 1$)	Программа выдает	Область обрабатывается верно
A	нет	-	-	-	нет
B					
C					
D					
E					

Заполним вторую область.$B \ni [-3;1]$, пусть $x = 0$, тогда проверим условия для if:

$$0 \geq -3$$

Условие выполняется

$$(0 \leq 9)$$

Условие выполняется

$$(0 > 1)$$

Условие не выполняется, программа выведет слово «принадлежит». Заполним таблицу для области B и для остальных.

Таблица 3.23

Область	Условие 1 $(x \geq -3)$	Условие 2 $(x \leq 9)$	Условие 3 $(x > 1)$	Программа выдает	Область обрабатывается верно
A	нет	-	-	-	нет
B	да	да	нет	принадлежит	да
C	да	да	да	не принадлежит	да
D	да	да	да	не принадлежит	нет
E	да	нет	-	-	нет

1) Именно так должна выглядеть правильно заполненная таблица.

2) Для правильной работы программы необходимо добавить проверку еще одного условия, при $x \geq 5$. И изменить 3 условие на

$$(x \leq 1)$$

.

Тогда условный оператор можно записать следующим образом:

$$if((x \geq -3)and(x \leq 1))or((x \geq 5)and(x \leq 9))$$

$$print(\text{"принадлежит"})$$

$$else$$

$$print(\text{"не принадлежит"})$$

Задача решена.

3.7 Задание № 25. Алгоритмы обработки массивов.

В задачах данного типа необходимо дописать часть кода. Все задачи связаны с массивами и имеют следующие вариации: поиск максимального и минимального элемента массива, подряд идущие пары элементов массива, анализ массива с накопителем и задачи связанные с другими алгоритмами. Для успешного решения таких задач и получения двух баллов, требуется хорошо знать следующие темы: одномерные и двумерные массивы.

Рассмотрим первый тип задач, связанный с подряд идущими элементами массива.

Задача №1. Дан целочисленный массив из 40 элементов. Элементы массива могут принимать целые значения от 0 до 10 000 включительно. Опишите на естественном языке или на одном из языков программирования алгоритм, позволяющий найти и вывести количество пар элементов массива, в которых десятичная запись хотя бы одного числа оканчивается на 2. В данной задаче под парой подразумевается два подряд идущих элемента массива.

Например, для массива из пяти элементов: 16 3 142 55 22 – ответ: 3.

Исходные данные объявлены так, как показано ниже на примерах для некоторых языков программирования и естественного языка. Запрещается использовать переменные, не описанные ниже, но разрешается не использовать некоторые из описанных переменных.

Python

```
//допускается также использовать
//две целочисленные переменные j и k
a = []
n = 40
for i in range(0, n):
a.append(int(input()))
...
```

Рис. 3.32

Решение. Необходимо написать программу, выводящую на экран количество пар чисел, из которых хотя бы одно оканчивалось на 2. Пара элементов, это два подряд идущих числа. Для того что бы определить на какую цифру заканчивается число, необходимо разделить это число на основание системы счисления, в которой записано само число и взять остаток. У нас десятичная запись числа, поэтому будем делить на 10.

Так как нам необходимо в результате вывести количество пар, то необходимо завести переменную – счетчик. Так же необходимо сделать проверку для пар чисел. Если одно из чисел заканчивается на 2 или если второе число из пары заканчивается на 2, то

данная пара удовлетворяет нашему условию и в переменную счетчик прибавляется единица.

Если в программе используется счетчик, то его необходимо изначально обнулить:

$$k = 0$$

Пары чисел будем проверять с помощью цикла for

$$for\ i\ in\ range(0, n-1):$$

Для проверки будем использовать условный оператор:

$$if\ (a[i]\%10 == 2)or(a[i+1]\%10 == 2):$$

$$k+ = 1$$

Так же необходимо добавить процедуру вывода

$$print(k)$$

Данный код позволит определить и вывести на экран количество пар чисел, из которых хотя бы одно оканчивается на 2:

```
k=0
for i in range(0,n-1):
    if (a[i]  %10==2)or (a[i+1]%10==2):
      k+=1
print(k)
```

Задача решена.

Рассмотрим аналогичный пример.

Задача № 2. Дан целочисленный массив из 50 элементов. Элементы массива могут принимать целые значения от 0 до 10 000 включительно. Опишите на естественном языке или на одном из языков программирования алгоритм, позволяющий найти и вывести количество пар элементов массива, в которых оба числа двузначные. В данной задаче под парой подразумевается два подряд идущих элемента массива.

Например, для массива из пяти элементов: 16 2 14 91 21 — ответ: 2. Исходные данные объявлены так, как показано ниже на примерах для некоторых языков программирования и естественного языка. Запрещается использовать переменные, не описанные ниже, но разрешается не использовать некоторые из описанных переменных.

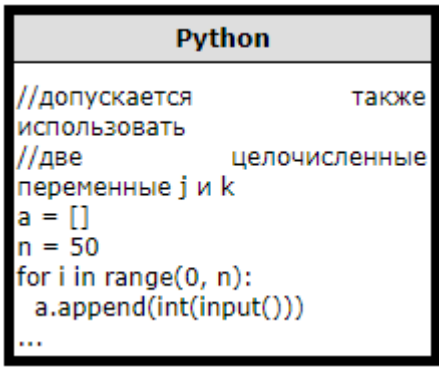


Рис. 3.33

Решение. Создадим счетчик и обнулим его:

$$k = 0$$

Для проверки каждой пары воспользуемся циклом for:

$$for\ i\ in\ range(0, n-1):$$

Все двузначные числа находятся в промежутке между 10 и 99, поэтому зададим условный оператор следующим образом:

$$if(a\,[i] \geq 10) and (a\,[i] \leq 99) and (a\,[i+1] \geq 10) and (a\,[i+1] \leq 99:$$

$$k+ = 1$$

Так же необходимо добавить процедуру вывода

$$print\,(k)$$

Итоговый код для проверки пары чисел на двузначность с последующим выводом их количества будет выглядеть следующим образом:

$$k = 0$$

$$for\ i\ in\ range(0, n-1):$$

$$if\ (a\,[i] \geq 10)\ and\ (a\,[i] \leq 99)\ and\ (a\,[i+1] \geq 10)\ and (a\,[i+1] \leq 99:$$

$$k+ = 1$$

$$print(k)$$

Задача решена.

Рассмотрим второй тип задач связанный с анализом массива и накопителем.

Задача № 3. Дан целочисленный массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 10 000 включительно. Опишите на одном из языков программирования алгоритм, который находит сумму элементов массива, меньших 200 и при этом кратных 5, а затем заменяет каждый такой элемент на число, равное найденной сумме. Гарантируется, что хотя бы один такой элемент в массиве есть. В качестве результата необходимо вывести изменённый массив, каждый элемент выводится с новой строчки.

Python

```
# допускается также
# использовать две
# целочисленные
# переменные j, k
a = []
n = 30
for i in range(0, n):
    a.append(int(input()))
...
```

Рис. 3.34

Решение. Необходимо написать такой алгоритм, который находит сумму элементов массива меньших 200 и при этом кратных 5. После чего происходит замена каждого такого элемента на число, равное найденной сумме.

Пусть в переменной k будет храниться сумма, тогда изначально обнулим ее:

$$k = 0$$

Для проверки элементов массива на выполнение условия воспользуемся циклом for:

$$for\ i\ in\ range(0, n):$$

Тогда в условном операторе мы будем выполнять следующую проверку:

$$if(a[i] < 200) and (a[i]\%5 == 0):$$

Тогда в переменной k происходит суммирование таких элементов массива:

$$k+ = a[i]$$

После проверки всех элементов массива, необходимо заменить каждый найденный элемент на полученную сумму. Для этого создадим еще один цикл for:

$$for\ i\ in\ range(0, n):$$

Замену можно произвести так:

$$if(a[i] < 200)and(a[i]\%5 == 0):$$

$$a[i] = k$$

Для вывода элементов массива, в котором каждый элемент вводится с новой строки, необходимо добавить процедуру:

$$print(a[i])$$

Задача решена.

Рассмотрим задачу, связанную с поиском минимального элемента массива.

Задача № 4. Дан целочисленный массив из 30 элементов. Элементы массива могут принимать натуральные значения от 1 до 10 000 включительно. Опишите на одном из языков программирования алгоритм, который находит минимум среди элементов массива, не делящихся нацело на 6, а затем заменяет каждый элемент, не делящийся нацело на 6, на число, равное найденному минимуму. Гарантируется, что хотя бы один такой элемент в массиве есть. В качестве результата необходимо вывести изменённый массив, каждый элемент выводится с новой строчки.

Python

```
# допускается также
# использовать две
# целочисленные переменные j и k
a = []
n = 30
for i in range(0, n):
    a.append(int(input()))
...
```

Рис. 3.35

Необходимо написать алгоритм, который находит минимальный элемент массива не кратный 6, а затем заменяет каждый такой элемент на найденное минимальное значение.

Пусть в переменной k хранится минимальное значение. Тогда изначально в эту переменную мы должны передать самое большое из возможных значений.

$$k = 10001$$

Для проверки элементов массива на выполнение условия воспользуемся циклом for:

$$for\ i\ in\ range(0, n):$$

Для проверки кратности запишем в условный оператор следующее:

$$if a[i]\%6! = 0\ and\ a[i] < k:$$

$$k = a[i]$$

После нахождения такого элемента, необходимо вновь пробежаться по элементам массива. Если текущий элемент массива не кратен 6, то в него записывается минимальное значение:

$$for\ i\ in\ range(0, n):$$

$$if a[i]\%6! = 0:$$

$$a[i] = k$$

Так же необходимо добавить процедуру вывода:

$$print(a[i])$$

Задача решена.

Рассмотрим заключительную задачу в данном блоке.

Задача № 5.

16) Дан целочисленный массив из 30 элементов. Элементы массива могут принимать целые значения от 0 до 10000 включительно. Опишите на одном из языков программирования алгоритм, который находит количество четных элементов массива, превышающих первый элемент, а затем заменяет каждый такой элемент на число, равное найденной количеству. Гарантируется, что хотя бы один такой элемент в массиве есть. В качестве результата необходимо вывести изменённый массив, каждый элемент выводится с новой строчки.

Например, для исходного массива из шести элементов: 4 11 12 24 2 16

программа должна вывести следующий массив 4 11 3 3 2 3

```
# допускается также
# использовать две
# целочисленные переменные j и k
a = [ ]
n = 30
for i in range(0, n):
   a.append(int(input()))
...
```

Рис. 3.36

Необходимо написать алгоритм, который находит количество четных элементов массива, превышающих первый элемент, а затем заменяет каждый такой элемент на число равное найденному количеству.

Для решения задачи нам необходимо воспользоваться 2 циклами for. Первый цикл будет, находить количество, второй изменять элементы и выводить на экран результат.

Пусть переменная k будет копить в себе количество четных элементов массива.

$$k = 0$$

$$for\ i\ in\ range(0, n):$$

$$if a[i] > a[0]\ and\ a[i]\%2 == 0:$$

$$k = k + 1$$

$$for\ i\ in\ range(0, n):$$

$$if a[i] > a[0]\ and\ a[i]\%2 == 0:$$

$$a[i] = k$$

$$print(a[i])$$

Задача решена.

Все задания № 25 из ЕГЭ схожи между собой, поэтому можно составить следующий алгоритм их выполнения:

1. Присвоение переменной в зависимости от алгоритма и ее обнуление.
2. Обход элементов массива в цикле for
3. Запись условия в условном операторе
4. Вывод результата

Литература

[1] Приказ Министерства образования и науки Российской федерации от 17 мая 2012 г. № 413 «Об утверждении федерального государственного образовательного стандарт среднего (полного) общего образования» // «Российская газета» — Федеральный выпуск № 5812 от 21.07.2012

[2] Крылов С.С., Методические рекомендации для учителей, подготовленные на основе анализа типичных ошибок участников ЕГЭ 2018 года по информатике и ИКТ [Текст]/С.С. Крылов- Москва 2018

[3] Любанович Б., Простой Python. Современный стиль программирования [Текст]/ Любанович Билл. — СПб.: Питер, 2016. — 480 с.: ил. — (Серия «Бестселлеры O'Reilly»).

[4] Майк мГ., Python программирование для начинающих [Текст]/ Майк МакГрат - Москва 2018

[5] Ройтберг М.А., Информатика и ИКТ. Подготовка к ЕГЭ в 2019 году. Диагностические работы [Текст]/ Я.Н. Зайдельман, М.А. Ройтберг – М.: Издательство МЦНМО, 2019

[6] Большой энциклопедический словарь [Электронный ресурс] URL: https://www.vedu.ru/bigencdic/ (дата обращения 24.11.2018)

[7] Интернет-ресурс. Требования к уровню подготовки обучающихся - Рабочая программа по информатике и икт основного общего образования [Электронный ресурс] URL: http://pochit.ru/informatika/5947/index.html?page=2 (дата обращения 14.02.19)

[8] Интернет-ресурс. Решу ЕГЭ [Электронный ресурс] Гущин Д. Д., 2011—2019. URL: https://inf-ege.sdamgia.ru (дата обращения 1.03.19)

[9] Интернет-ресурс. Питонтьютор [Электронный ресурс] 2012–2017. URL: https://pythontutor.ru (дата обращения 4.04.19)

[10] Интернет-ресурс. kpolyakov.spb.ru [Электронный ресурс] 2000-2019 К. Поляков. URL: https://www.kpolyakov.spb.ru/school/ege.htm (дата обращения 24.12.18)