«Мир информатики»
Журнал для тех, у кого информатика – любимый школьный предмет
Выпуск № 90, июнь 2024 г.

01100010001010100100111110010001100100100010100100100101 1

Школа программирования

Язык программирования Python. Варианты действий в программе

Приведем программу на языке Python решения задачи расчета площади любого прямоугольника:

```
a = int(input('Введите длину прямоугольника в см '))
b = int(input('Введите высоту прямоугольника в см '))
#Расчет площади
pl = a * b
#Вывод ответа на экран
print('Площадь прямоугольника равна', pl, 'кв. см')
```

В ней обязательно должны выполняться все указанные действия (ввод двух размеров прямоугольника, расчет площади и вывод ответа на экран). В то же время в программах возможны ситуации, когда требуется выполнение не строго определенного действия, а одного из двух или более вариантов действия, при этом выбор того или иного варианта зависит от некоторого условия. Возможны также ситуации, когда некоторое действие должно выполняться не всегда, а только при определенном условии.

Рассмотрим особенности каждого случая.

1. Два варианта действий

Пример. Дано целое число. Определить, является ли оно четным.

Решение

Начальная часть программы решения задачи:

```
n = int(input('Введите целое число '))
```

Далее возможны два варианта действий (два варианта инструкции `print()` вывода информации на экран):

1)
```
print('Это число четное')
```

2)
```
print('Это число нечетное')
```

В таких случаях (возможны два варианта действий) в программе необходимо использовать инструкцию **`if`** в следующей форме:

```
if <условие>:
    <Действия 1-го варианта (1-я серия инструкций)>
else:
    <Действия 2-го варианта (2-я серия инструкций)>
```

(Обратите внимание на отступы во 2-й и 4-й строках – инструкция **if** является составной.

В самом простом случае *<условие>* – это два арифметических выражения между которыми записан знак операции сравнения[1].

В языке Python есть 6 операций сравнения:

Таблица 1

	Знак операции	Означает
1	<	Меньше
2	<=	Меньше либо равно
3	>	Больше
4	>=	Больше либо равно
5	==	Равно
6	!=	Не равно

Примеры:

```
a < 0
x <= a + b
(a + b)/2 > 0
2 * c >= 7 * d - 1
N % 5 = 3
v != 100
```

Каждое условие может соблюдаться (быть истинным, например таким является условие `a < 0` при значении переменной `a`, равном –12) или не соблюдаться (быть ложным), и после выполнения в программе операции сравнения получается соответствующий результат логического типа (**True** или **False**).

Приведенный вариант инструкции **if** называют «полный вариант».

Схема работы такого варианта показана на рис. 1[2]:

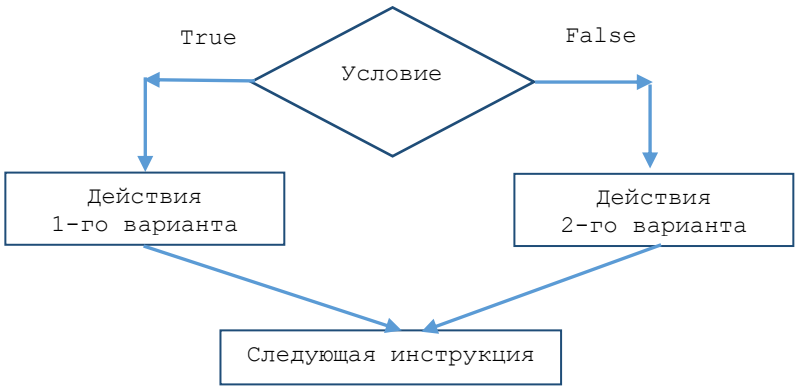


Рис. 1

[1] Операции сравнения называют также «операциями отношения», а знаки этих операций – «операторами сравнения» или «операторами отношения».

[2] Приведенную схему называют «развилка» или «ветвление», а части инструкции **if** и **else** – «ветви».

Применительно к рассмотренному примеру инструкция **if** оформляется так:

```
if n % 2 == 0:
    print('Это число четное')
else:
    print('Это число нечетное')
```

Возникает вопрос: «А можем ли мы считать первым вариантом действий вывод на экран сообщения о том, что заданное число – нечетное?» Конечно, можем:

```
if … :
    print('Это число нечетное')
else:
    print('Это число четное')
```

Но при этом условие, записываемое после служебного слова **if**, должно быть уточнено – оно должно быть *истинным* для случая выполнения в программе *действий первого варианта* (см. рис. 1), то есть таким – `n % 2 == 1`:

```
if n % 2 == 1:
    print('Это число нечетное')
else:
    print('Это число четное')
```

Самостоятельно убедитесь в том, что для обоих способов оформления инструкции результат, согласно схеме на рис. 1, будет одним и тем же как для четных, так и для нечетных значений числа `n`.

Как видно из схемы на рис. 1 и согласно общей форме полного варианта инструкции `if` (см. выше), как в ветви **if**, так и в ветви **else** может быть указана не одна инструкция, а их серия.

Пример. Даны два вещественных числа `a` и `b`. Если первое больше второго, то увеличить каждое число в 2 раза, иначе – уменьшить в два раза.

Соответствующая программа:

```
a = float(input('a = '))
b = float (input('b = '))
if a > b:
    a = a * 2
    b = b * 2
else:
    a = a/2
    b = b/2
print('a =', a)
print('b =', b)
```

Контрольные вопросы

1. В каких случаях в программе используется полный вариант инструкции `if`? Как он оформляется? Как он «работает»? (Что происходит при его выполнении?) Нарисуйте графическую схему выполнения.

2. Что представляет из себя условие, записываемое в инструкции `if` в простейшем случае? Какие знаки операций сравнения (отношения) могут использоваться в нем?

3. Что является результатом операции сравнения?

Задания

1. Определите (не оформляя программу) значение переменной `c` после выполнения следующего фрагмента программы:

```
a = 40
b = 10
```

```
b = -a/2 * b
if a < b:
    c = b - a
else:
    c = a - 2 * b
```

2. Определите значение переменной `a` после выполнения следующего фрагмента алгоритма[3]:

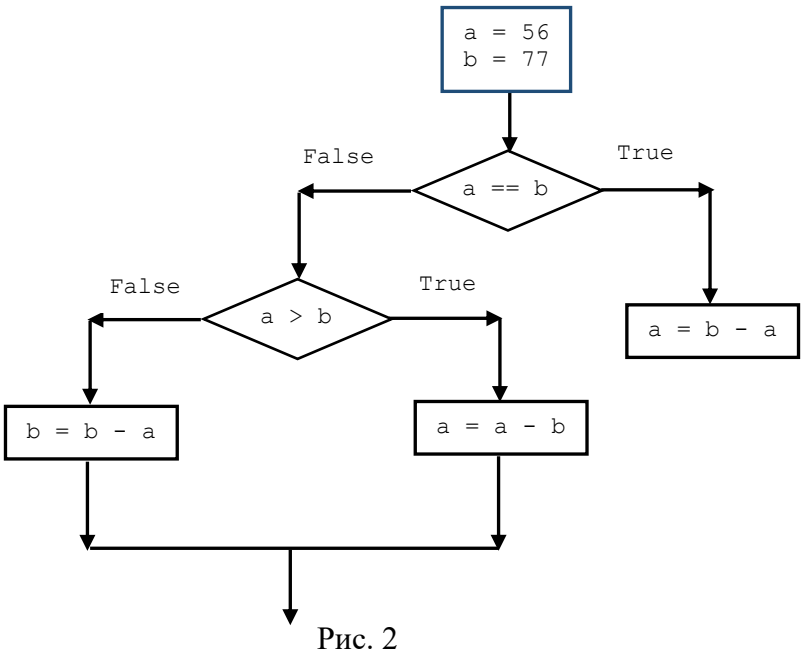


Рис. 2

Задания на разработку программ

1. Даны два различных вещественных числа. Напишите программу, которая определяет:

а) какое из них больше;

б) какое из них меньше.

2. Напишите программу, которая определяет, является ли число `a` делителем числа `n`.

3. Напишите программу, которая определяет, в какую из областей – **I** или **II** (рис. 3) – попадает точка с заданными координатами. Для простоты принять, что точка не попадает на границу областей.

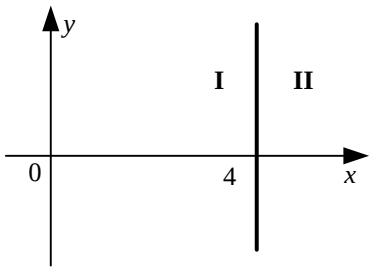


Рис. 3

4. Напишите программу, в которой рассчитывается значение `y` при заданном значении `x`:

$$y = \begin{cases} \sin^2 x & \text{при } x > 0, \\ 1 - 2\sin x^2 & \text{в противном случае.} \end{cases}$$

5. Даны радиус круга и сторона квадрата. Напишите программу, которая определяет, у какой фигуры площадь больше.

---

[3] Одним из способов записи алгоритма является оформление так называемых «блок-схем». На рис. 2 приведен фрагмент блок-схемы.

6. Напишите программу, которая решает задачу: «Дано целое число `n`. Вывести на экран следующее за ним четное число».

7. Даны коэффициенты `a`, `b` и `c` квадратного уравнения $ax^2 + bx + c = 0$ (`a != 0`). Напишите программу, которая определяет, имеет ли это уравнение корни или нет (сами корни, если они есть, вычислять не нужно).

В инструкции **`if`** возможно также использование так называемых «сложных условий» – состоящих из двух или нескольких простых условий[4], соединенных служебными словами (логическими операторами) **`and`** (и), **`or`** (или) или **`not`** (не).

Пример. Предположим, что компания набирает сотрудников, возраст которых – от 25 до 50 лет включительно. Нужно написать программу, которая запрашивает возраст претендента и выдает ответ: «Подходит» или «Не подходит» он по этому признаку.

Пусть возраст сотрудника задан и записан в переменной `vozr`. Тогда фрагмент программы, в котором выводится ответ, будет выглядеть так:

```
if vozr >= 25 and vozr <= 50:
    print('Подходит')
else:
    print('Не подходит')
```

При проверке сложного условия сначала выполняются операции сравнения (<, <=, >, >=, ==, !=), а затем – логические операции в таком порядке: сначала все операции с оператором `not`, затем – с оператором `and`, и в самом конце – с оператором `or` (во всех случаях – слева направо). Для изменения порядка действий используют круглые скобки.

Результат выполнения логических операций с операторами **`and`** и **`or`** над двумя простыми условиями `C1` и `C2` определяется по правилам, представленным в табл. 2 и 3:

1) **`and`**

Таблица 2

C1	C2	C1 **and** C2
False	False	False
False	True	False
True	False	False
True	True	True

то есть результат будет истинным (**`True`**) только в случае, когда оба простых условия `C1` и `C2` истинны;

2) **`or`**

Таблица 3

C1	C2	C1 **or** C2
False	False	False
False	True	True
True	False	True
True	True	True

то есть результат будет истинным в случае, когда хотя бы одно простое условие является истинным.

[4] Простым условимся называть условие, особенности которого указаны перед табл. 5.

Для задачи о возрасте сотрудника возможные варианты приведены в табл. 4:

Таблица 4

vozr	vozr >= 25	vozr <= 50	vozr >= 25 **and** vozr <= 50	Результат (согласно рис. 1)
23	**False**	**True**	**False**	Не подходит
26	**True**	**True**	**True**	Подходит
56	**True**	**False**	**False**	Не подходит

Результат выполнения логической операции с оператором `not` над условием `C` определяется так:

C	**not** C
False	**True**
True	**False**

то есть он противоположен значению условия `C`.

Иногда условия получаются достаточно длинными, и их хочется перенести на следующую строку. Сделать это в Python можно двумя способами:

– использовать обратный слэш («\»):

```
if v < 400 and v != 2 and v != 3 and v != 12 and \
v != 13 and v != 22 and v != 23:
    …
```

или взять все условие в скобки (перенос внутри скобок разрешен):

```
if (v < 400 and v != 2 and v != 3 and v != 12 and v != 13
and v != 22 and v != 23):
    …
```

В языке Python разрешены двойные неравенства, например:

```
if A < B < C:
    …
```

означает то же самое, что и

```
if A < B and B < C:
    …
```

В качестве условия в инструкции `if` можно использовать также:

1) логические функции, то есть функции, возвращающие результат логического типа:

```
n = int(input('Введите целое число '))
if chet(n):
    print('Это число четное')
else:
    print('Это число нечетное')
```

где `chet()` – функция, возвращающая результат **`True`**, если ее параметр (значение, указанное в скобках) является четным числом, и **`False`** – в противном случае;

2) оператор **`in`** (оператор проверки принадлежности), который проверяет, принадлежит ли некоторый объект (число, символ, переменная и т. п.) набору значений (списку, строке, диапазону чисел и т. п.):

```
a = 3
if a in range(10):  #Если значение переменной a
                    #входит в диапазон значений,
                    #возвращаемый функцией range() (см. ниже)
    …
sim = input('Введите символ ')
s = input('Введите строку символов ')
if sim in s:        #Если символ sim имеется в строке s
    …
```

```
raduga = […]
zvet = 'Зеленый'
if zvet in raduga:      #Если значение переменной zvet
                        #имеется в списке raduga
    …
```

3) операторы **is**/**is not** (операторы проверки идентичности), которые определяют, ссылаются ли (или не ссылаются) две переменные на один и тот же объект.

Конечно, можно комбинировать простые (или сложные) условия с логическими функциями, операторами **in** и **is**/**is not**, а также с логическими константами **True** и **False**. Примеры приведены в дополнении.

Контрольные вопросы

1. Что такое сложное условие? Какие логические операции могут использоваться в нем?
2. Каков порядок (приоритет) выполнения логических операций? Как изменить этот порядок?
3. Что может быть использовано в инструкции `if`, кроме простых и сложных условий?

Задачи для разработки программ

1. Дано натуральное число. Определить, является ли оно двузначным.
2. Даны два целых числа. Определить, является ли хотя бы одно из них делителем другого.
3. Даны координаты точки на плоскости. Определить, попадает ли точка в область **I** (рис. 4). Для простоты принять, что координаты точки не равны соответствующим границам этой области.

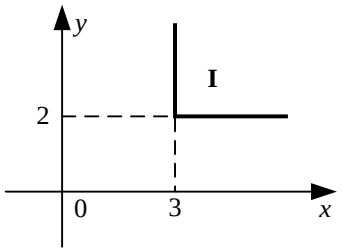


Рис. 4

4. Определить, является ли запись заданного четырехзначного числа симметричной.
5. Определить, является ли треугольник со сторонами `a`, `b`, `c`:

а) равносторонним;

б) равнобедренным.

6. Дано трехзначное число. Определить, входит ли в него цифра 6.
7. Определить, войдет ли в конверт с внутренними размерами `a` и `b` мм прямоугольная открытка с размерами `c` и `d` мм. Для размещения открытки в конверте необходим зазор в 1 мм с каждой стороны.

2. Один, но не обязательный вариант действий

Пример. Даны три целых числа, среди которых есть отрицательные. Вывести на экран отрицательные числа на одной строке.

Используем в программе три переменные – `a`, `b` и `c`.

Начальная часть программы решения задачи:

```
print('Введите 3 целых числа (как минимум одно - отрицательное)')
a = int(input())
b = int(input())
```

```
c = int(input())
print('Среди них отрицательные:')
```

Далее – в каком случае выводить значение первого числа `a`? Только если соблюдается условие `a < 0`. Аналогично и для двух других чисел. В таких случаях (когда какие-то действия в программе выполняются не всегда, а только при определенном условии) используется инструкция `if` в следующей форме:

```
if <условие>:
    <Действия (серия инструкций)>
```

Такой вариант инструкции `if` называют «неполный вариант».

Применительно к рассмотренному примеру завершающая часть программы решения задачи оформляется так:

```
if a < 0:
    print(a, ' ', , end = '')
if b < 0:
    print(b, ' ', , end = '')
if c < 0:
    print(c)
```

Схема работы неполного варианта инструкции **`if`** показана на рис. 5[5]:

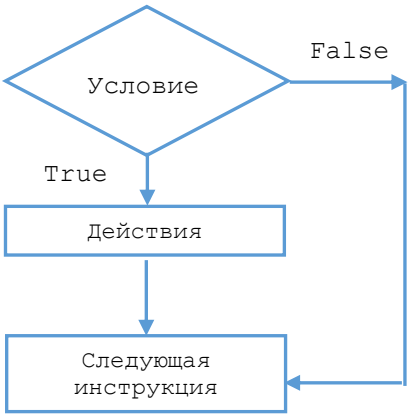


Рис. 5

Еще раз обратим внимание на то, что нет альтернативных вариантов действий, как в случае, рассмотренном в пункте 5, а есть единственно возможный, но не обязательный, вариант.

Контрольные вопросы

В каких случаях в программе используется неполный вариант инструкции `if`? Как он оформляется? Как он «работает»? (Что происходит при его выполнении?) Нарисуйте графическую схему выполнения.

Задачи для разработки программ

1. Даны три целых числа. Вывести на экран те из них, которые являются четными.
2. Дано вещественное число. Вывести на экран его абсолютную величину (условно принимая, что соответствующей стандартной функции Python нет). Полный вариант инструкции `if` не использовать.
3. Даны два различных вещественных числа. Определить наибольшее и наименьшее из них, не используя полный вариант инструкции `if`, а применив:

а) два неполных варианта инструкции;

б) один неполный вариант инструкции.

---

[5] Приведенную схему называют «обход».

4. Даны три вещественных числа. Вывести на экран те из них, которые принадлежат интервалу [1.6, 3.8].

5. Даны три числа a, b, c. Определить, сколько из них положительных.

Решение

Можно после ввода исходных данных:

```
print('Введите 3 числа')
a = float(input())
b = float(input())
c = float(input())
```

рассмотреть все возможные варианты, используя неполные варианты инструкции if:

```
if a <=0 and b <= 0 and c <= 0:
    k = 0
if (a > 0 and b <= 0 and c <= 0 or b > 0 and a <= 0 and c <= 0
       or c > 0 and a <= 0 and b <= 0):
    k = 1
if (a > 0 and b > 0 and c <= 0 or a > 0 and c > 0 and b <= 0
       or b > 0 and c > 0 and a <= 0):
    k = 2
if a > 0 and b > 0 and c > 0:
    k = 3
#Вывод ответа
…
```

Обратим внимание на то, что, поскольку логическая операция с оператором and выполняется раньше, чем операция с оператором or, то скобки в сложном условии записывать не надо (имеющиеся в программе скобки позволяют записать длинное сложное условие на двух строках).

6. Даны шесть целых чисел a, b, c, d, e, f. Определить, сколько из них положительных.

Решение

Если решать задачу так, как предыдущую, то программа получится достаточно громоздкой (а если бы заданных чисел было 10?). Можно сократить размер программы, учитывая следующие рассуждения.

Во-первых, будем вводить заданные числа по одному и сразу будем проверять, является ли очередное число положительным (в предыдущей программе проверка проводилась после ввода всех трех чисел).

Во-вторых, используем в программе переменную k, которая будет хранить[6] количество положительных чисел среди уже рассмотренных заданных чисел.

Составим таблицу:

Число	Значение	k
a	7	1
b	–2	1
…		k
n	6	?
…		

Как определить количество положительных чисел после ввода некоторого очередного положительного числа n, если информацию о таком количестве среди уже рассмотренных чисел хранит величина k? Ответ – надо к «старому» значению количества k прибавить 1:

```
k = k + 1
```

Напомним, что в программировании такая запись возможна.

[6] Напомним, что переменные используются в программах для хранения информации.

При сделанных рассуждениях вся программа решения задачи может быть оформлена так:

```
k = 0  #Начальное значение переменной k
a = int(input('Введите значение числа a '))
if a > 0:
    k = k + 1      #Применен неполный вариант инструкции if
b = int(input('Введите значение числа b '))
if b > 0:
    k = k + 1
c = int(input('Введите значение числа c '))
if c > 0:
    k = k + 1
d = int(input('Введите значение числа d '))
if d > 0:
    k = k + 1
e = int(input('Введите значение числа e '))
if e > 0:
    k = k + 1
f = int(input('Введите значение числа f '))
if f > 0:
    k = k + 1
#Вывод ответа
…
```

Согласитесь, что приведенный вариант программы не только имеет меньший размер, чем вариант с полным перебором значений, но и понятнее.

Переменные, аналогичные переменной `k`, которые предназначены для подсчета количества некоторых значений, называют «счетчиками».

7. Даны три целых числа `a`, `b`, `c`. Определить сумму положительных из них.

8. Даны шесть целых чисел `a`, `b`, `c`, `d`, `e`, `f`. Определить сумму положительных из них.

Комментарий к решению

Если предыдущую задачу можно решить полным перебором, как это делалось при решении задачи 5, то в данном случае это нерационально (программа получится достаточно громоздкой).

Лучше использовать методику, аналогичную использованной при решении задачи 6.

Используем в программе переменную `sum`, которая будет хранить сумму положительных чисел среди уже рассмотренных заданных чисел.

Составим таблицу:

Число	Значение	`sum`
`a`	7	7
`b`	2	9
…		`sum`
`n`	6	?
…		

Как определить сумму положительных чисел после ввода некоторого очередного положительного числа `n`, если информацию о такой сумме среди уже рассмотренных чисел хранит величина `sum`? Ответ – надо к «старому» значению суммы `sum` прибавить `n`:

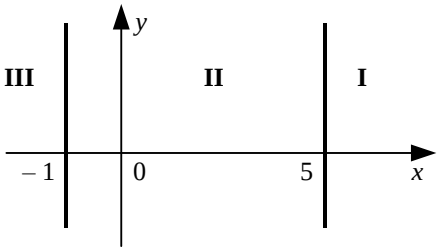
```
sum = sum + n
```

Переменную `sum`, накапливающую в себе сумму нескольких значений, будем называть «сумматором»[7].

Предлагаем читателям обратить внимание на переменную-счетчик и переменную-сумматор, так как они неоднократно будут использоваться в программах далее.

3. Три и более вариантов действий

Рассмотрим задачу: «На плоскости выделены три зоны (I, II, III).



Дана координата `x` точки. Определить, в какую зону попала эта точка. Принять, что `x` не равно границам зон (–1 и 5)».

Решение

Можно в программе использовать 3 неполных варианта инструкции **`if`** (без ветви **`else`**):

```
if x < -1:
    print('В зону III')
if x > 5:
    print('В зону I')
if x > -1 and x < 5:
    print('В зону II')
```

Можно ли «сэкономить» одно слово **`if`** и использовать полный вариант инструкции?[8]

```
if x < -1:
    print('В зону III')
if x > 5:
    print('В зону I')
else:
    print('В зону II')
```

Проверим.

При `x` == 10 ответ будет правильным (см. схему на рис. 1).

При `x` == 0 – то же.

При `x` == –8 на экран будет выведено (почему – установите самостоятельно):

```
В зону III
В зону II
```

то есть ответ будет ошибочным.

Правильный вариант:

```
if x < -1:
    print('В зону III')
else:  #Возможны 2 варианта действия, то есть надо использовать
       #полный вариант инструкции if
    if x > 5:
        print('В зону I')
```

---

[7] В компьютере *сумматор* – один из элементов процессора, в котором происходит сложение двух двоичных чисел.

[8] Начинающие программисты, как правило, отвечают: «Да, можно». Так ли это – см. далее.

```
    else:
        print('В зону II')
```

Получается, что одна инструкция **if** «вложена» в другую, поэтому такой вариант оформления называют «вложенной инструкцией **if**».

Можно «объединить» первое служебное слово **else** и следующее за ним слово **if** и записать слово **elif**:

```
if x < -1:
    print('В зону III')
elif x > 5:
    print('В зону I')
else:
    print('В зону II')
```

Использование служебного слова **elif** позволяет упростить запись и избежать чрезмерных отступов.

Ветвь `elif` может присутствовать несколько раз; наличие ветви **else** необязательно:

```
if …:
    …
elif …:
    …
elif …:
    …
elif …:
    …
```

Если в цепочке `if-elif-elif-…` истинными являются несколько условий, то «срабатывает» первое из них. Например, программа:

```
if cost < 1000:
    print ('Скидок нет')
elif cost < 2000:
    print('Скидка 2%')
elif cost < 5000:
    print('Скидка 5%')
else:
    print('Скидка 10%')
```

при `cost == 1500` выводит на экран:

```
Скидка 2%
```

хотя условие `cost < 5000` тоже выполняется.

Это означает, что в общем случае:

```
if <условие 1>:
    <действия 1>
elif <условие 2>:
    <действия 2>
elif <условие 3>:
    <действия 3>
elif …:
    …
elif <условие n>:
    <действия n>
else:
    <действия (n + 1)>
```

схема работы вложенной инструкции **if** будет такой:

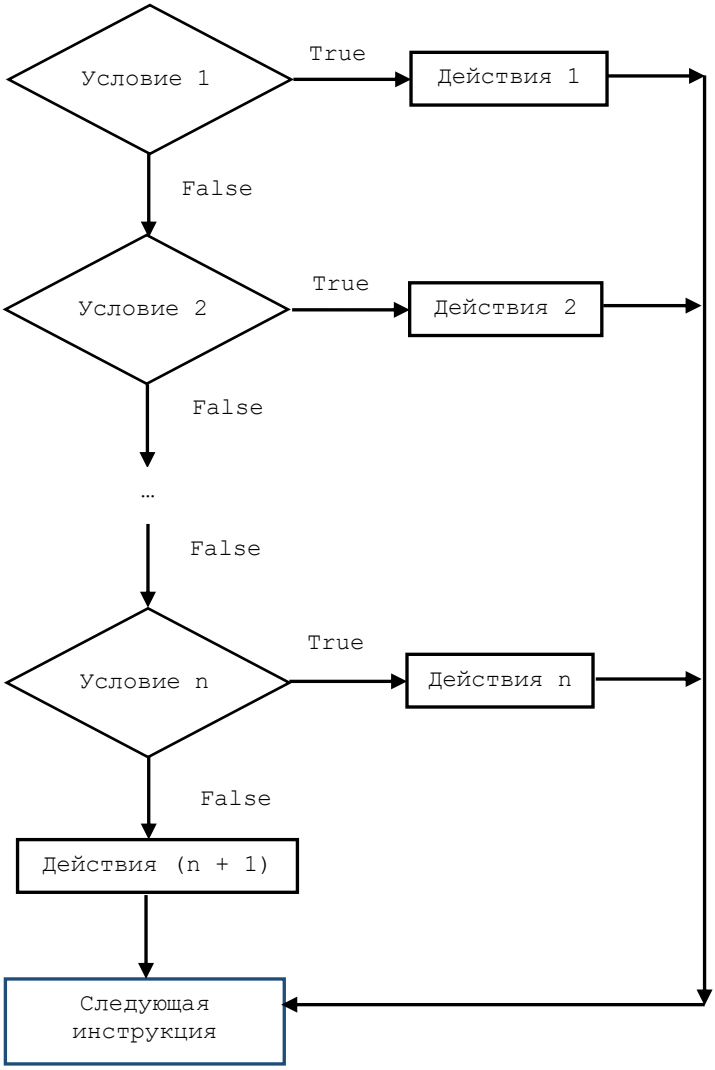


Рис 6

Контрольные вопросы

В каких случаях в программе используется вложенная инструкция **`if`**? Как она оформляется? Как она «работает»? (Что происходит при ее выполнении?)

Задачи для разработки программ[9]

1. Известны возрасты Мити и Васи. Определить, кто из них старше или они одного возраста.

2. Известен вес боксера-любителя (в кг, в виде вещественного числа). Известно, что вес таков, что боксер может быть отнесен к одной из трех весовых категорий:

1) легкий вес – до 60 кг;

2) первый полусредний вес – до 64 кг;

3) полусредний вес – до 69 кг.

Определить, в какой категории будет выступать данный боксер.

3. Даны два различных целых числа `a` и `b` ($a \neq 0$, $b \neq 0$). Определить, является ли `a` делителем `b`, или `b` является ли делителем `a`, или ни одно из чисел не является делителем другого.

4. Дан порядковый номер дня недели (1, 2, ..., 7). Вывести на экран его название (понедельник, вторник, ..., воскресенье).

5. Дан порядковый номер месяца (1, 2, ..., 12). Вывести на экран его название (январь, февраль, ..., декабрь).

[9] Во всех программах использовать одну инструкцию **`if`**.

6. Дан порядковый номер месяца (1, 2, ..., 12). Вывести на экран количество дней в этом месяце. Принять, что год не является високосным. В инструкции `if` использовать не более трех ветвей.

В заключение рассмотрим задачу, которая часто встречается в программировании: «Определить максимальное значение из двух заданных различных вещественных чисел».

Решение

Переменным величинам в программе дадим имена `a`, `b` (заданные числа) и `max` (максимальное из них).

Так как возможны два варианта ответа, то в программе следует использовать полный вариант инструкции **`if`**:

```
…
if a > b:
    max = a
else:
    max = b
…
```

Можно также фрагмент, связанный с определением значения `max`, записать короче:

```
max = a
if b > a:
    max = b
```

или

```
max = b
if a > b:
    max = a
```

Теперь «откроем небольшой секрет» (☺). В Python есть стандартная функция `max()`, которая возвращает максимальное значение среди указанных для нее аргументов[10]. Она вызывается так:

```
max = max(a, b, …)
```

Несмотря на это, варианты определения максимального значения, приведенные до этого, полезны, так как они могут быть использованы для решения других аналогичных задач.

Дополнение

Выше говорилось о том, что в записи инструкции **`if`** (во всех рассмотренных вариантах) в качестве условия можно писать любые комбинации простых и сложные условий, логических функций, операторов **`in`** и **`is`**/**`is not`**, а также логических констант **`True`** и **`False`**:

```
x > 0 and x < 100 and even(z) or n > 1000
True and i <= 20
```

и т. п.

Подобные записи называют «логическими выражениями». Логические выражения можно использовать для присваивания значений величинам логического типа (**`bool`**):

```
b = diskr > 0
b2 = x > 0 and x < 100 and even(z) or n > 1000
```

[10] Есть также аналогичная функция `min()`, которая возвращает минимальное из двух или нескольких значений.

Такие инструкции называют «инструкциями логического присваивания». Они работают следующим образом:

1) вычисляется результат логического выражения в правой части (по правилам, описанным применительно к сложным условиям в пункте 1);

2) этот результат (**`True`** или **`False`**) присваивается переменной, указанной в левой части инструкции.

Переменные типа `bool` также можно записывать в инструкции `if`:

```
d = discr > 0
if d == True:[11]
    …
```

и в инструкции **`while`** (см. следующий раздел).

Их можно «объединять» с другими простыми и сложными условиями:

```
if d == True and a != 0:     #Или if d and a != 0:
```

и т. п.

Язык программирования Python. Повторение действий в программе[12]

Как правило, в программах приходится организовывать повторение одних и тех же действий. Для этого используются инструкции **`for`** и **`while`**[13].

Рассмотрим их особенности.

1. Инструкция `for`

Она применяется в тех случаях, когда в программе какие-то действия (инструкции) повторяются и при этом некоторая величина меняется с постоянным шагом[14].

Пример 1. Для вывода «столбиком» всех целых чисел от 10 до 20 без использования инструкции **`for`** в программе потребовалось бы записать:

```
print(10)
print(11)
...
print(20)
```

Видно, что есть повторяющиеся действия (вывод на экран числа) и что при этом выводимое число меняется с шагом, равным 1. Можно применить инструкцию **`for`**.

Общий вид этой инструкции:

```
for <параметр инструкции> in <набор значений>:
    <тело инструкции>        #Записывается с отступом
```

где *<тело инструкции>* – действия, повторяющиеся при работе инструкции (это могут быть любые инструкции Python); *<параметр инструкции>* – имя величины, меняющейся при

---

[11] Обратим внимание, что более короткая запись:

```
if d:
    …
```

также возможна.

[12] Многократное выполнение одинаковых действий в программировании называют «цикл». См. также далее о термине «итерация».

[13] В Python эти инструкции называют так же, как в других языках программирования, – «цикл **`for`**» и «цикл **`while`**».

[14] Это утверждение далее будет уточнено.

повторении действий; <*набор значений*> – набор значений параметра инструкции, для которых проводится повторение.

В рассмотренном примере параметром инструкции является выводимое число, а телом инструкции является одна другая инструкция – `print()`.

Так как выводимое число меняется от 10 до 20 с шагом 1, в качестве <*набор значений*> можно использовать функцию `range()`. Эта функция будет очень часто использоваться далее, поэтому остановимся на ней подробно.

Функция `range()` возвращает набор целых чисел, образующих арифметическую прогрессию. Например, если нужно получить числа 0, 1, 2, …, 6, то следует так оформить вызов функции:

```
range(7)
```
[15]

а в общем случае – для получения `n` последовательных целых чисел, начиная с 0:

```
range(n)
```

Часто нужно, чтобы набор из `n` последовательных целых чисел начинался с 1. В таких случаях функция оформляется так:

```
range(1, n + 1)
```

Обратим внимание на то, что в последнем случае интервал чисел задается двумя значениями – начальным и конечным, причем конечное значение не используется.

В принципе, первый член прогрессии может быть любым целым числом (в том числе отрицательным). Если он равен `a`, а последний член прогрессии – `b`, то при `b > a` функция `range()` должна быть оформлена так:

```
range(a, b + 1)
```

а если `b < a`, то так:

```
range(a, b - 1, -1)
```
[16]

Примечание. В последнем случае указан шаг изменения значений – он равен –1, а в первом по умолчанию шаг принят равным 1.

Можно также получить набор чисел, шаг изменения которых отличается от 1 и –1 (но он тоже должен быть целым числом).

Вернемся к примеру о выводе чисел «столбиком». Если выводимое число обозначить именем `a`, то инструкция **`for`** оформляется следующим образом:

```
for a in range(10, 21):
    print(a)
```

Такая запись означает, что в программе должны повторяться действия по выводу на экран числа `a`, которое будет принимать значения (с помощью функции `range()`) от 10 до 20.

С другой стороны, при разработке программ важно уметь увидеть, какие действия повторяются и какая величина при этом меняется.

Пример 2. Пусть нужно рассчитать и вывести «столбиком» квадраты всех целых чисел от 50 до 60.

Без использования инструкции **`for`** в программе потребовалось бы записать:

```
kv = 50 * 50          #Или kv = 50 ** 2
print(kv)
kv = 51 * 51          #Или kv = 51 ** 2
```

---

[15] Обратите внимание – записывается число 7, хотя нужно получить максимальное число 6.

[16] Обратите внимание – записывается число `b - 1`, хотя нужно получить минимальное число, равное `b`.

```
print(kv)
…
kv = 60 * 60            #Или kv = 60 ** 2
print(kv)
```

Видно, что есть повторяющиеся действия (расчет и вывод на экран квадрата числа) и что при этом выводимое число меняется от 50 до 60 с шагом, равным 1. Значит, тоже можно применить инструкцию **for** с функцией `range()`:

```
for n in range(50, 61):
    kv = n * n      #Или kv = n ** 2
    print(kv)
```

Можно также оформить этот фрагмент короче:

```
for n in range(50, 61):
    print(n * n)    #Или     print(n ** 2)
```

В теле инструкции **for** можно записывать любые другие инструкции, например **if**:

```
for a in range(101):
    if a % 10 == 5:
        print(a)
```

Какая задача решается в приведенной программе, установите самостоятельно.

Еще пример: «Определить сумму всех чисел от 1 до 100».

Для решения надо каждое из указанных чисел прибавить к ранее рассчитанной сумме, используя для хранения суммы переменную-сумматор (см. выше):

```
sum = 0 #Начальное значение переменной-сумматора
for n in range(1, 101):
    sum = sum + n
print(sum)
```

Схема работы инструкции **for** с числовым параметром и при использовании функции `range()` с шагом 1 показана на рис. 1:

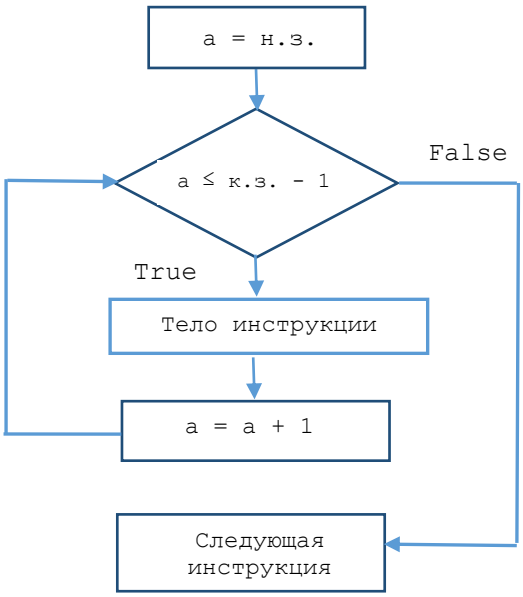


Рис. 1

где `a` – параметр инструкции, `н.з.` – начальное значение параметра (в функции `range()`), `к.з.` – конечное значение параметра (в функции `range()`).

Можно определить, что количество повторений тела инструкции для рассматриваемого случая равно:

(к.з. – 1) – н.з.+ 1 = к.з.– н.з.

Поэтому говорят, что при использовании инструкции **for** и функции `range()` количество повторений тела цикла известно. С другой стороны, когда известно количество повторений каких-то действий, в программе удобно использовать инструкцию **for**.

Здесь же заметим, что повторений действий в программировании часто называют «итерацией» (от латинского *iteratio* – повторение). Кроме того, «итерацией» называют также каждое *отдельное* повторение тела цикла.

В качестве набора значений параметра инструкции **for**, для которых проводится повторение каких-то действий, могут быть использованы также:

• произвольный набор значений (в том числе разного типа, хотя такое встречается редко):

```
for a in 10, 12, 18, 'один', 'два', 'три':  #Для всех
                                            #перечисленных значений
    print(a)
```

• отдельные символы величины строкового типа:

Программа	Результат ее выполнения

`s = 'ЛОКОМОТИВ'`
`for sim in s: #Для каждого символа величины s`
`  print(sim)`
	Л
	О
	К
	О
	М
	О
	Т
	И
	В

• элементы так называемого списка:

Программа	Результат ее выполнения

`m = [1, 2, 3, 4, 5]`
`for n in m: #Для каждого числа в списке m`
`  print(n * n)`
	1
	4
	9
	16
	25

и строки текстового файла.

Учитывая это, можем сделать важное уточнение – инструкция **for** применяется в программе, когда какие-то действия повторяются *для некоторого набора значений* (ряда чисел, отдельных символов, строк и др.)[17].

Контрольные вопросы

1. Что такое «цикл»?

2. В каких случаях используют инструкцию **for**? Какой ее общий вид?

3. Что такое «параметр инструкции **for**»? Что такое «тело инструкции»? Что может быть использовано в качестве параметра инструкции **for**?

4. Как должна быть оформлена функция `range()`, если действия в теле инструкции **for** должны выполняться для всех значений:

а) от 0 до `n` (`n` > 0) с шагом, равным 1;

б) от 4 до `k` (`k` > 4) с шагом, равным 1;

в) от 19 до 0 с шагом, равным –1;

[17] Поэтому саму инструкцию иногда называют «инструкция `for` … **in**».

г) от 18 до 8 с шагом, равным –1.

5. Может ли тело инструкции **for** с функцией `range()` при шаге изменения параметра, равном 1, не выполниться ни разу?

6. Сколько раз будет выполняться тело инструкции **for** со следующим «заголовком»:

- `for a in range(1, 10):`
- `for b in range(10, 20):`
- `for i in range(n, m):  #(m ≥ n)`

Задания

Определите:

а) значение параметра `a` после окончания работы следующей инструкции:

```
for a in range(10, 21):
    print(a)
```

б) значение параметра `n` после окончания работы следующей инструкции[18]:

```
m = [1, 2, 3, 4, 5]
for n in m:
    print(n)
```

Это будет важный результат, который надо учитывать при разработке программ.

Заметим также, что во многих языках программирования запрещено или не рекомендуется в теле инструкции цикла менять значение ее параметра. В программах на Python это допустимо, хотя необходимость в этом встречается крайне редко.

Задачи для разработки программ

1. Напечатать «столбиком» кубы всех целых чисел от 10 до `b` (значение `b` вводится с клавиатуры; `b` ≥ 10).

2. Напечатать таблицу соответствия между массой в фунтах и массой в килограммах для значений 1, 2, ..., 10 фунтов (1 фунт = 453 г) в виде:

```
Фунты Килограммы
    1         …
    2
    …
   10
```

3. Напечатать таблицу умножения на 7:

```
1 x 7 = 7
2 x 7 = 14
...
9 x 7 = 63
```

4. Напечатать все нечетные числа из интервала [10, 100]. Разработать два варианта программы:

1) с использованием инструкции **if**;

2) без использования этой инструкции.

5. Напечатать все нечетные двузначные числа, у которых последняя цифра равна 3 или 7.

6. Напечатать все целые числа от `a` до `b`, кратные некоторому числу `c`.

7. Напечатать все двузначные числа, сумма квадратов цифр которых делится на 13.

8. Найти сумму $1^2 + 2^2 + 3^2 + \ldots + n^2$ при заданном значении n.

Рекомендации по выполнению. Используйте переменную-сумматор (см. выше).

9. При заданном значении `n` ≥ 2 вычислить сумму $1 \times 2 + 2 \times 3 + ... + (n - 1) \times n$.

[18] В приведенном фрагменте программы использован так называемый «список»..

10. Определить сумму всех четных трехзначных чисел.

11. Определить количество трехзначных чисел, сумма цифр которых равна некоторому значению `s`.

Рекомендации по выполнению. Используйте переменную-счетчик.

12. Определить количество всех трехзначных чисел, кратных семи и у которых сумма цифр также кратна семи.

Напишите также программу, которая вычисляет факториал заданного целого положительного числа `n`, то есть произведение чисел от 1 до `n` (1 × 2 × 3 × ... × `n`).

Частным случаем инструкции **`for`** является вариант, при котором величина – параметр в теле инструкции не используется. Пусть, например, требуется повторить какие-то действия 10 раз (например, вывести приветствие «Привет!»). С учетом схемы на рис. 1 соответствующий фрагмент может быть записан в виде:

```
for z in range(10):
    print('Привет!')
```

или

```
for z in range(1, 11):
    print('Привет!')
```

или другим подобным способом.

Видно, что величина `z` в теле инструкции **`for`** не используется – при выполнении инструкции она играет роль счетчика числа повторений.

Другим распространенным примером являются программы решения задач, в которых надо ввести с клавиатуры, а затем обработать значения нескольких чисел (например, определить сумму чисел, максимальное/минимальное число или его порядковый номер и т. п.). В них используется следующий фрагмент:

```
for nomer in range(n):  #n – общее количество чисел
    a = float(input('Введите очередное число '))
    …    #Обработка заданного значения числа a
```

Здесь также величина `nomer` в теле инструкции **`for`** не используется.

Задачи для разработки программ

1. Напечатать ряд чисел 20 в виде:

```
20 20 20 20 20 20 20 20 20 20
```

2. Вывести на экран любое заданное число любое заданное число раз в виде, аналогичном показанному в предыдущей задаче.

3. Дано целое число `n` и вещественное число `a`. Найти их произведение, не используя операцию умножения.

4. Даны вещественное число `a` и целое число `n`. Вычислить a^n, не используя операцию возведения в степень.

5. Одноклеточная амеба каждые 3 часа делится на 2 клетки. Определить, сколько клеток будет через 3, 6, 9, ..., 24 часа, если первоначально была одна амеба.

6. Начав тренировки, лыжник в первый день пробежал 10 км. Каждый следующий день он увеличивал пробег на 10% от пробега предыдущего дня. Определить пробег лыжника во второй, третий, ..., десятый день тренировок.

Другие задания на разработку программ

1. Последовательность Фибоначчи образуется так: первый и второй члены последовательности равны 1, каждый следующий равен сумме двух предыдущих (1, 1, 2, 3, 5, 8, 13, ...). Составьте программу для нахождения 3-го, 4-го, …, 10-го членов последовательности Фибоначчи.

Комментарии к выполнению

Используем в программе следующие основные величины:

`ocher` – очередной рассчитываемый член последовательности;

`pred` – член последовательности, предшествующий очередному члену;

`predpred` – член последовательности, предшествующий члену `пред`.

Сначала имеем:

```
pred = 1
predpred = 1
```

Затем рассчитываем очередной (третий) элемент:

```
ocher = pred + predpred
```

после чего «готовимся» к расчету следующего элемента. Для этого изменяем значения `pred` и `predpred`. Какой из двух вариантов «подготовки» правильный:

```
predpred = pred
pred = ocher
```

или

```
pred = ocher
predpred = pred
```

– решите самостоятельно[19].

После этого определяем `ocher`:

```
ocher = pred + predpred
```

и т. д.

Можно использовать инструкцию **`for`**.

2. В приведенных только что фрагментах программ последние рассчитанные значения величин `pred` и `predpred` не используются. Составьте программу для нахождения `n`-го члена последовательности Фибоначчи, в которой этот недостаток будет устранен. Принять, что `n ≥ 4`.

Разработайте также программы для построения с помощью исполнителя «черепаха» следующих фигур:

а) квадрата;

б) правильного шестиугольника;

в) равностороннего треугольника;

г) правильного 12-угольника;

д) правильного 20-угольника.

Используйте инструкцию цикла **`for`**. Угол поворота «черепахи» в задачах б–д установите самостоятельно.

---

[19] В то же время, если для изменения значений `predpred` и `pred` использовать множественное присваивание, то порядок записи имен этих переменных не важен:

```
for k in range(…):
    ocher = pred + predpred
    predpred, pred = pred, ocher
```

или

```
for k in range(…):
    ocher = pred + predpred
    pred, predpred = ocher, pred
```

Инструкция **for** может быть использована в программе для обеспечения некоторой паузы[20], если в ее теле записать некоторые условные (неиспользуемые) инструкции. Например:

```
for z in range(40000000):
    a = 0
```

Продолжительность паузы будет зависеть от быстродействия компьютера и от конечного значения параметра инструкции.

Задание

Подберите на своем компьютере такое конечное значения параметра инструкции **for**, при котором в программе будет пауза длительностью примерно 5 сек.

***

Об инструкции (цикле) **while** будет рассказано в следующем выпуске журнала «Мир информатики»

[20] В Python имеется специальная функция `sleep()`, обеспечивающая приостановку выполнения программы на заданное число секунд (например, `sleep(3)`).

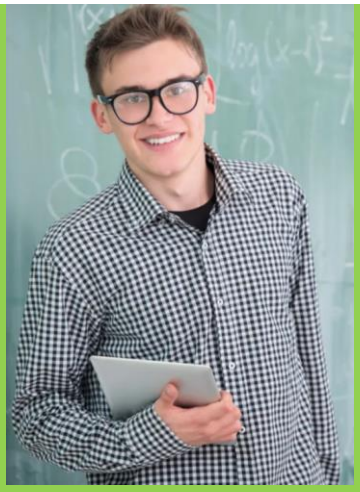
Доска почёта “Мира информатики”

На Доске почёта нашего журнала представлены учащиеся, принявшие активное участие в наших конкурсах и награжденные дипломами в текущем учебном году.

.

Ковальчук Иван
ученик 5-го класса школы № 11
г. Струнино Владимирской обл.
(учитель Волков Ю.П.)

Амосов Кирилл
средняя школа села Сердар,
Республика Марий Эл,
(учитель Чернова Л.И.)



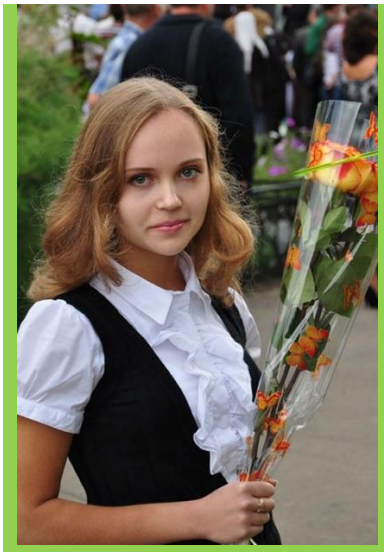
Бородин Антон
Москва, школа № 1530
«Школа Ломоносова»,
(учитель
Каменецкая Т.С.)

Вагизов Ислам
Республика Татарстан,
Мамадышский р-н,
совхоз “Мамадышский”,
политехнический колледж,
(преподаватель Порываева Н.С.)

Волобуева Ирина
Барнаульский государственный педагогический колледж имени В.К. Штильке
(преподаватель Лукьянова Н.В)

Гладков Родион
Ставропольский край,
г. Зеленокумск, школа № 1
(учитель Букина Е.Ю.)

Голубева Марина
средняя школа села Восточное Нижегородской обл.,
(учитель Долгова Г.А.)



Зубрицкая Мария
средняя школа села Сердар, Республика Марий Эл
(учитель Чернова Л.И.)

Лукьянов Кирилл
Барнаульский государственный педагогический колледж имени В.К. Штильке
(преподаватель Лукьянова Н.В)

Лёвина Татьяна
средняя школа поселка Осиновка, Алтайский край
(учитель Евдокимова А.И.)

Макарова Мария
средняя школа села Сердар,
Республика Марий Эл
(учитель Чернова Л.И.)

Молошникова Евгения
г. Пенза,
школа № 73
(учитель Диков А.В.)

Павленко Екатерина
Ставропольский край,
г. Зеленокумск, школа № 1
(учитель Букина Е.Ю.)

Плодущев Георгий
Ставропольский край,
г. Зеленокумск, школа № 1
(учитель Букина Е.Ю.)

Пусь Иван
г. Пенза,
школа № 73
(учитель Диков А.В.)

Снегирёв Максим
Республика Татарстан,
совхоз “Мамадышский”,
политехнический колледж,
(преподаватель Порываева Н.С.)

Судаков Владислав
средняя школа
поселка Осиновка,
Алтайский край

Чемизов Артур
г. Пенза,
школа № 73
(учитель Диков А.В.)

Фаворов Игорь
Москва, школа № 1530 «Школа Ломоносова»
(учитель Каменецкая Т.С.)

Гульназарова Ангелина
Ставропольский край,
г. Зеленокумск,
школа № 1
(учитель Букина Е.Ю.)

Шевченко Матвей
Ставропольский край,
г. Зеленокумск,
школа № 1
(учитель Букина Е.Ю.)

Смирнова Полина
Владимирская обл.,
г. Струнино,
школа № 11
(учитель Волков Ю.П.)

Спасибо, дорогие ребята!
Желаем вам дальнейших успехов!

Внимание! Конкурс!

Победителями конкурса № 66 также признаны и будут награждены дипломами:

— Актимежева Софья, Сидоренко Виктория и Сушко Валерия, Ставропольский край, г. Зеленокумск, школа № 1, учитель **Букина Е.Ю.**;

— Лукьянов Кирилл и Волобуева Ирина, Барнаульский государственный педагогический колледж имени Василия Константиновича Штильке, преподаватель **Лукьянова Н.В.**;

— Фаворов Игорь, Москва, школа № 1530 «Школа Ломоносова», учитель **Каменецкая Т.С.**