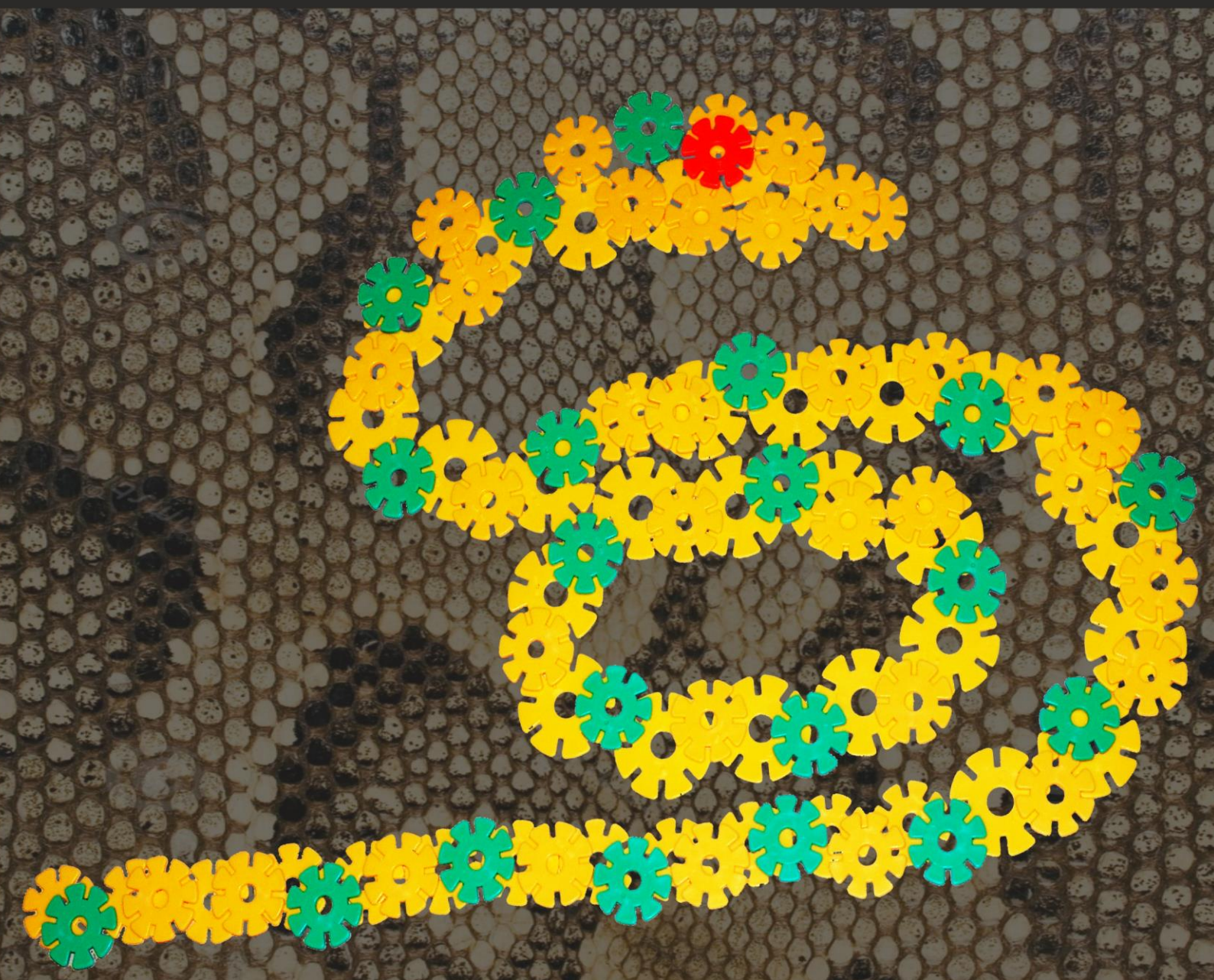
Рубанцев Валерий

Решебник на языке Python

к Электронному задачнику Абрамяна

•Begin •Integer •Boolean



RVGames.de

Валерий Рубанцев

Решебник на языке *Python* к *Электронному задачнику* Абрамяна

- Begin
- Integer
- Boolean

Бесплатное издание

Автор книги не несёт ответственности за возможный вред от использования информации, составляющей содержание книги и приложений.

От автора

В этой книге мы будем решать задачи из **Электронного задачника по программированию *Programming Taskbook* 4**. А именно из трёх первых наборов заданий: **Begin**, **Integer** и **Boolean**. Их названия говорят сами за себя: они адресуются начинающим программировать на *Питоне* и предназначены для изучения таких базовых понятий любых языков программирования, как:

- *Ввод и вывод данных*
- *Вычисления по формулам*
- *Типы данных*
- *Функции*
- *Структура простой программы*
- *Операторы и операции*
- *Арифметические операции*
- *Определение переменных*
- *Оператор присваивания*

Особое внимание уделяется вещественному типу **float**, целому типу **integer** и логическому типу **boolean**, а также операциям над ними:

- *Целочисленное деление* ***//***
- *Вещественное деление* ***/***
- *Деление по модулю (остаток от деления)* ***%***
- *Простые и сложные высказывания*
- *Логические выражения*
- *Операции сравнивания: = <> > < >= <=*
- *Логическое отрицание НЕ -* ***not***
- *Логическое И -* ***and***
- *Логическое ИЛИ -* ***or***
- *Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ -* ***xor***

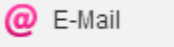
Электронный задачник по программированию разработан Михаилом Эдуардовичем Абрамяном в 1998-2015 годах. Он не только содержит наборы заданий, но и контролирует их выполнение. В случае ошибки вы получите необходимую информацию для её нахождения и исправления. Таким образом, даже начинающие программисты могут самостоятельно отрабатывать навыки программирования на языке *Питон*!

Если у вас возникнут затруднения (а они неизбежны на начальном этапе обучения), то в книге вы найдёте полное и подробное решение всех задач

из указанных выше наборов, а в папке **_Projects** - исходные коды всех задач. Обращайтесь к ним за помощью и поддержкой!

Разработка программ ведётся в удобной среде *PyCharm*, но вы можете с успехом решать задачи и в более простой среде *IDLE*.

Валерий Рубанцев

	***RVGames.de***
	***VRubantsev@t-online.de***
	***https://www.facebook.com/groups/1771696989786687/?ref=bookmarks***

Условные обозначения, принятые в книге:

> Дополнение или замечание

> Ненавязчивое требование или указание

```
Исходный код
```

Исходные коды всех проектов находятся в папке **_Projects**

Оглавление

Устанавливаем *Питона*

Самая необременительная процедура в программировании!

На официальном сайте www.python.org откройте меню *Downloads* и щёлкните по строчке **Windows** (Рис. 1).

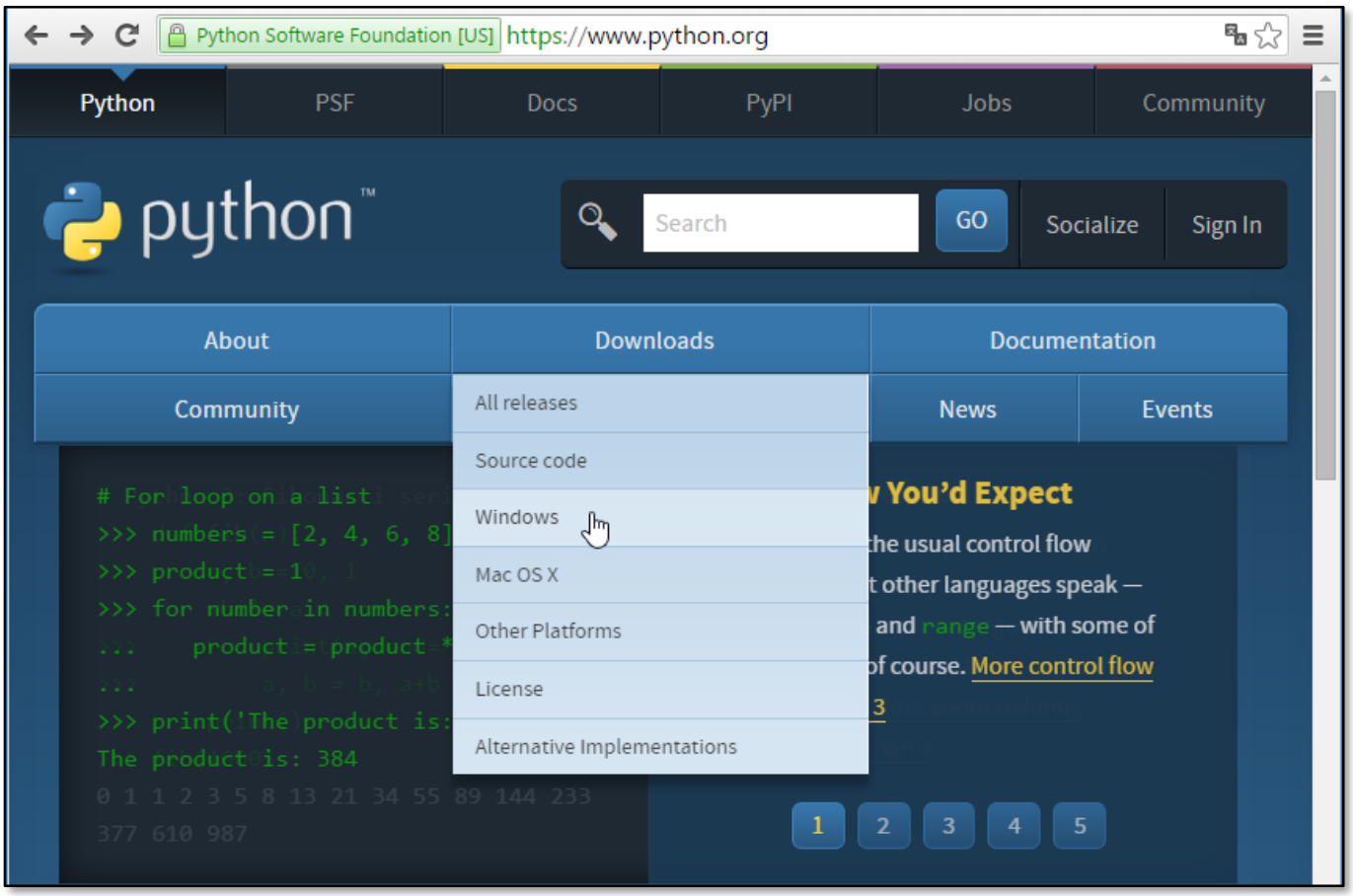


Рис. 1. Логово *Питона*

На следующей странице выберите нужный вам **установочный файл** для загрузки (Рис. 2).

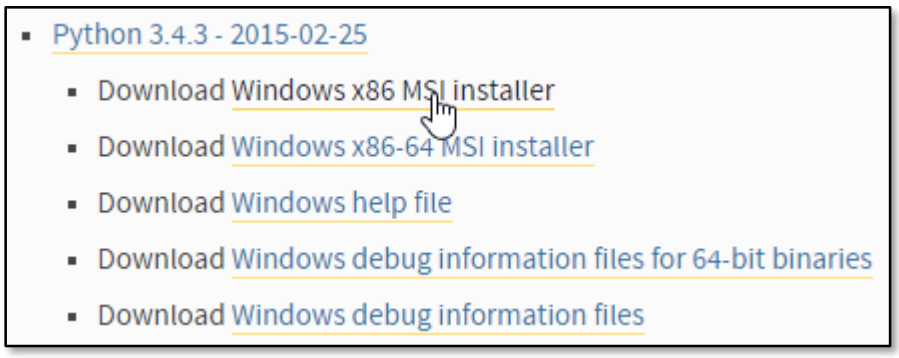


Рис. 2. Не самый свежий *Питон*

Используйте именно этот установочный файл!

Запустите файл **python-3.4.3.msi** (Рис. 3).

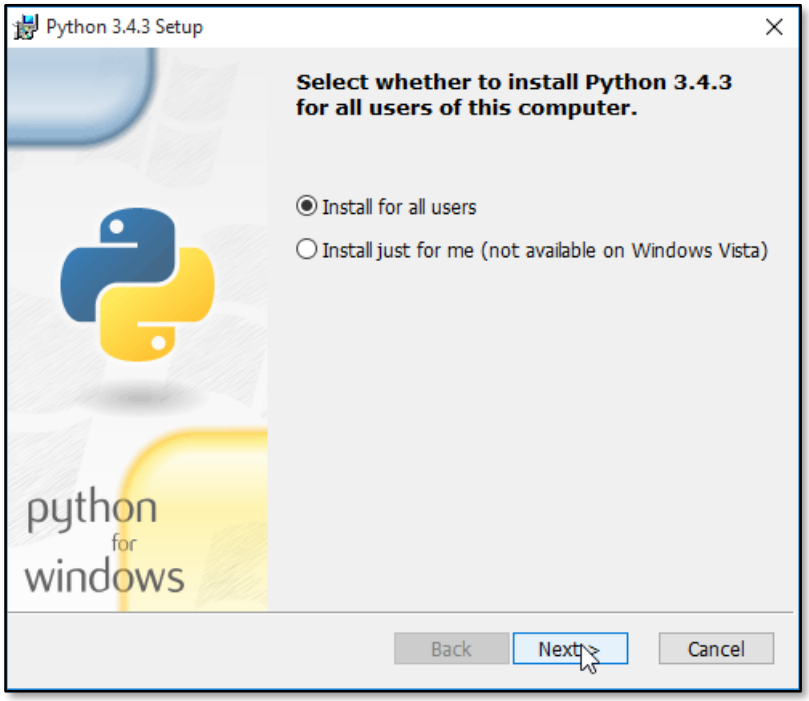
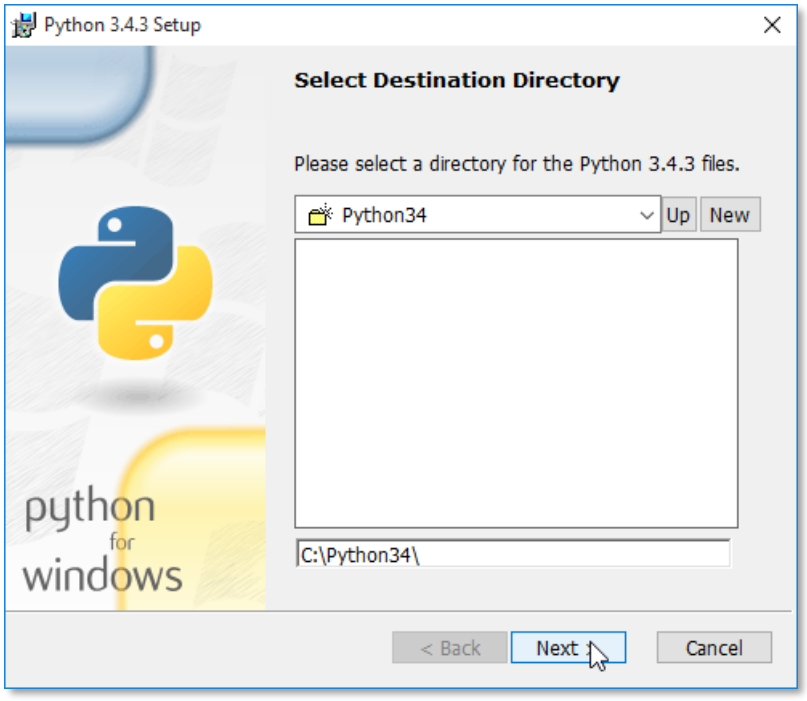


Рис. 3. *Питон* пошёл!

В следующих диалоговых окнах снова нажмите кнопку **Next** (Рис. 4).



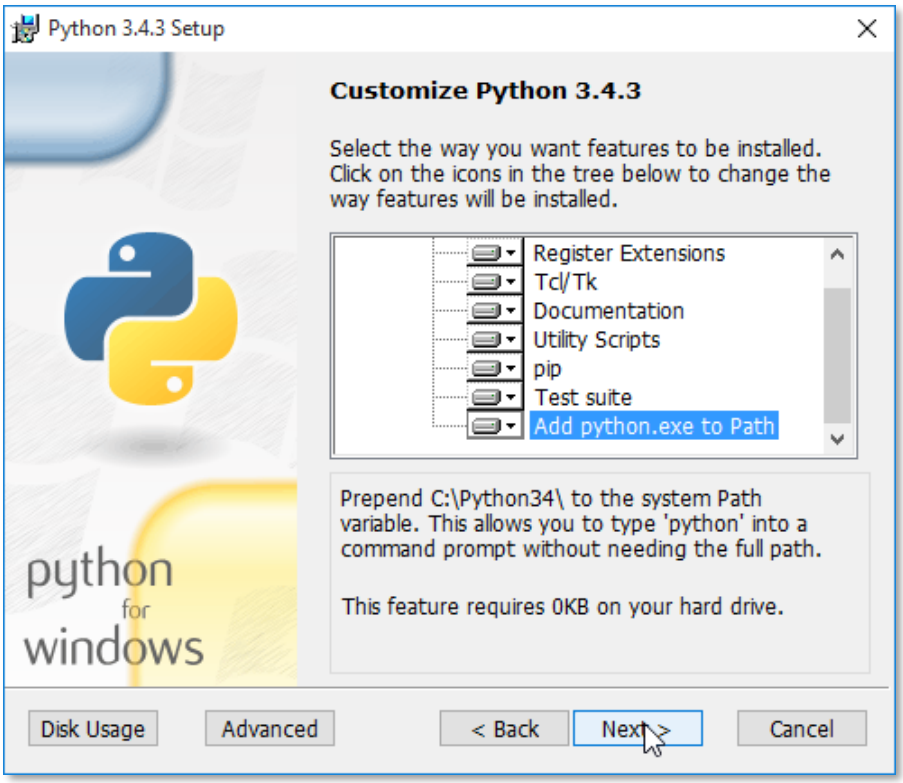


Рис. 4. Вслед *Питону*

Установочный процесс пошёл. За его ходом вы можете наблюдать в очередном диалоговом окне (Рис. 5).

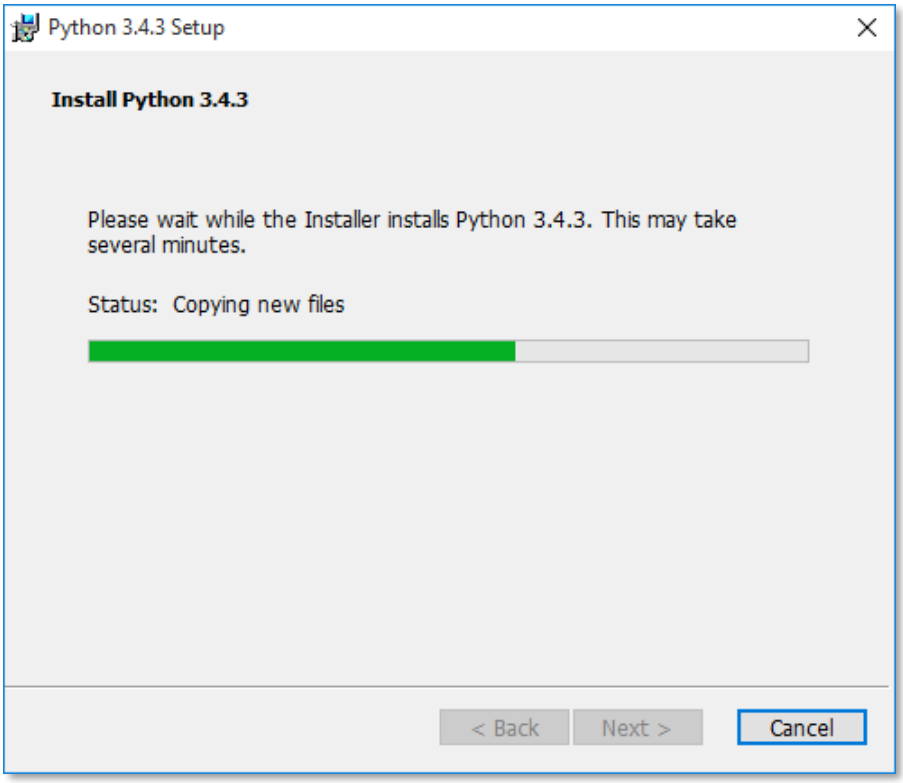


Рис. 5. *Питон* растёт…

Сколько *Питону* ни виться, а успешный конец ему будет (Рис. 6).

Рис. 6. *Установка закончена*

Создайте **ярлык** на рабочем столе для быстрого запуска среды разработки *IDLE* (Рис. 7).

Рис. 7. *Питоний ярлык*

Самые нетерпеливые уже могут начать программировать на *Питоне* (Рис. 8).

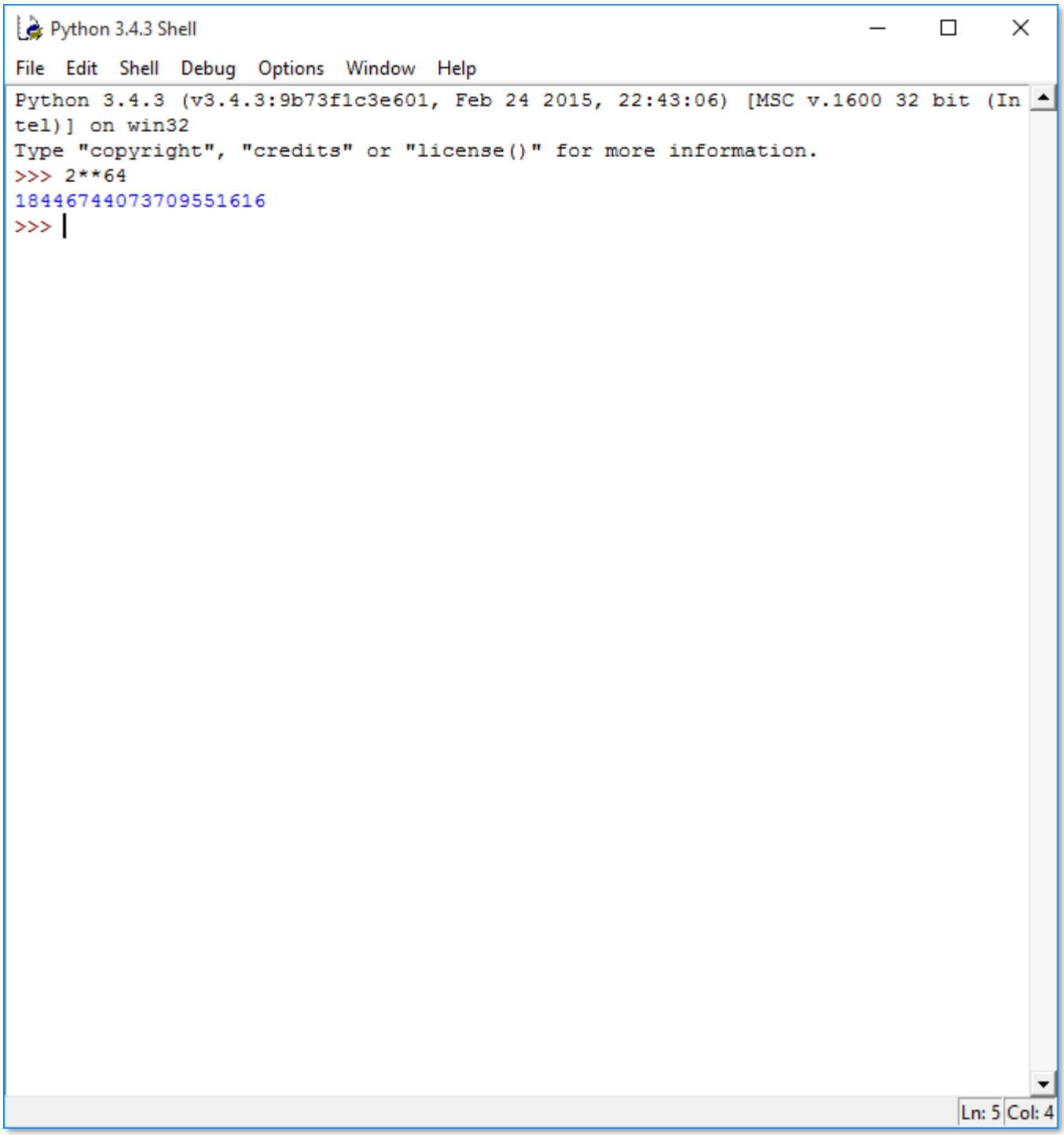


Рис. 8. *Знаменитая шахматная задача решена вмиг!*

Устанавливаем *PyCharm*

Если вы не склонны к полумерам, то не ограничивайте свои потребности скромными возможностями *IDLE*, а сразу установите более удобную и мощную среду разработки **PyCharm**.

На сайте http://www.jetbrains.com/pycharm/download/ нажмите кнопку **Download Community** для загрузки бесплатной, «коммунистической» версии программы (Рис. 1).

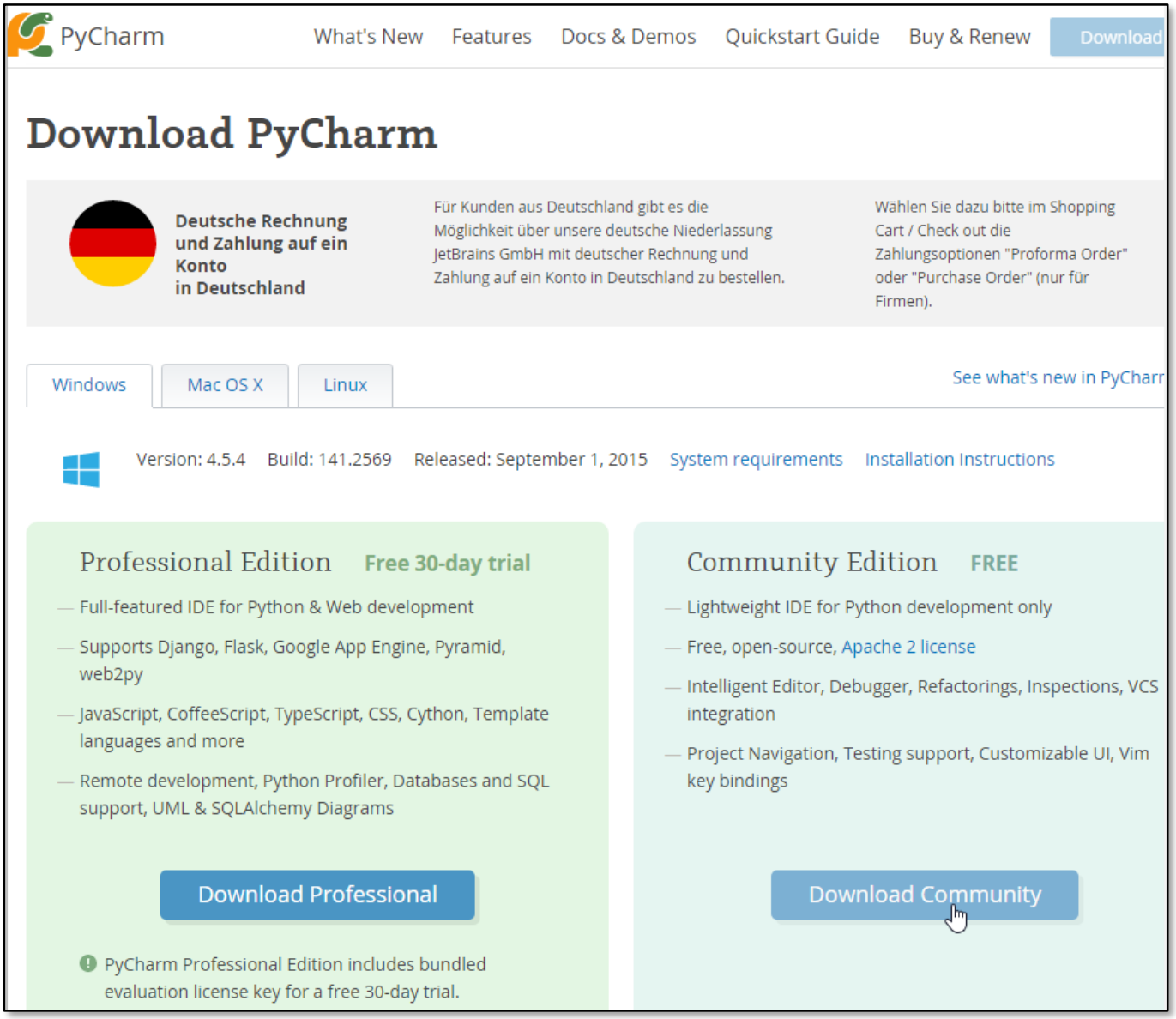


Рис. 1. Шарман!

Размер установочного файла 158 мегабайтов, поэтому время ожидания его появления в загрузочной папке зависит от скорости *Интернета* (Рис. 2).

Рис. 2. Качаем!

Программа довольно ёмкая, но сейчас они все такие (Рис. 3).

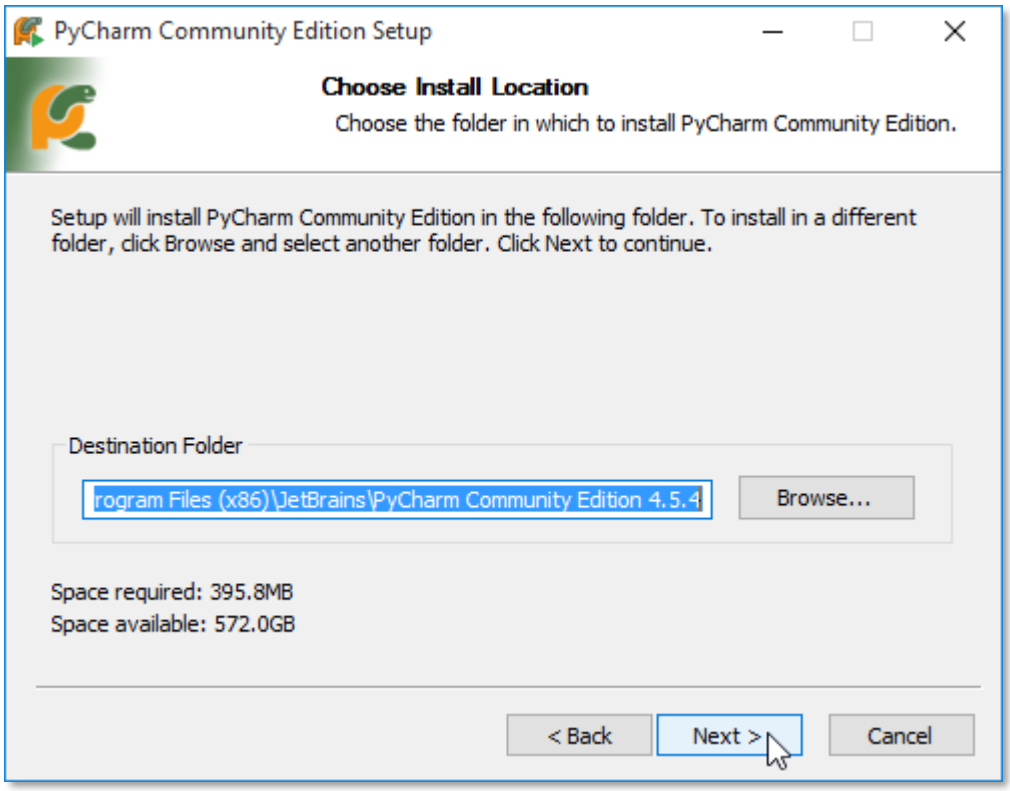


Рис. 3. Качаем!!

Ставим галочки и идём дальше (Рис. 4).

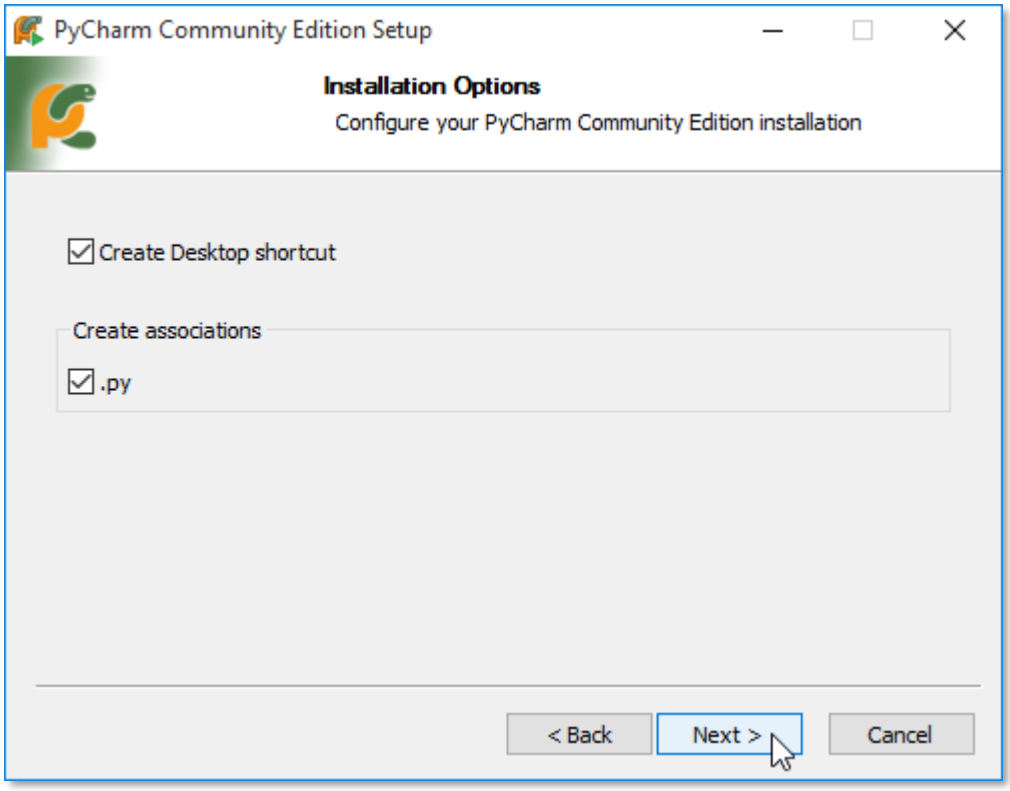


Рис. 4. Запускаем!

Долго ли, коротко ли, но программа установится, и её можно запустить (Рис. 5).

Рис. 5. Запустили

Понажимав малозначительные кнопочки, вы попадёте в *Окно приветствия*, в котором можно начать **новый проект** (Рис. 6).

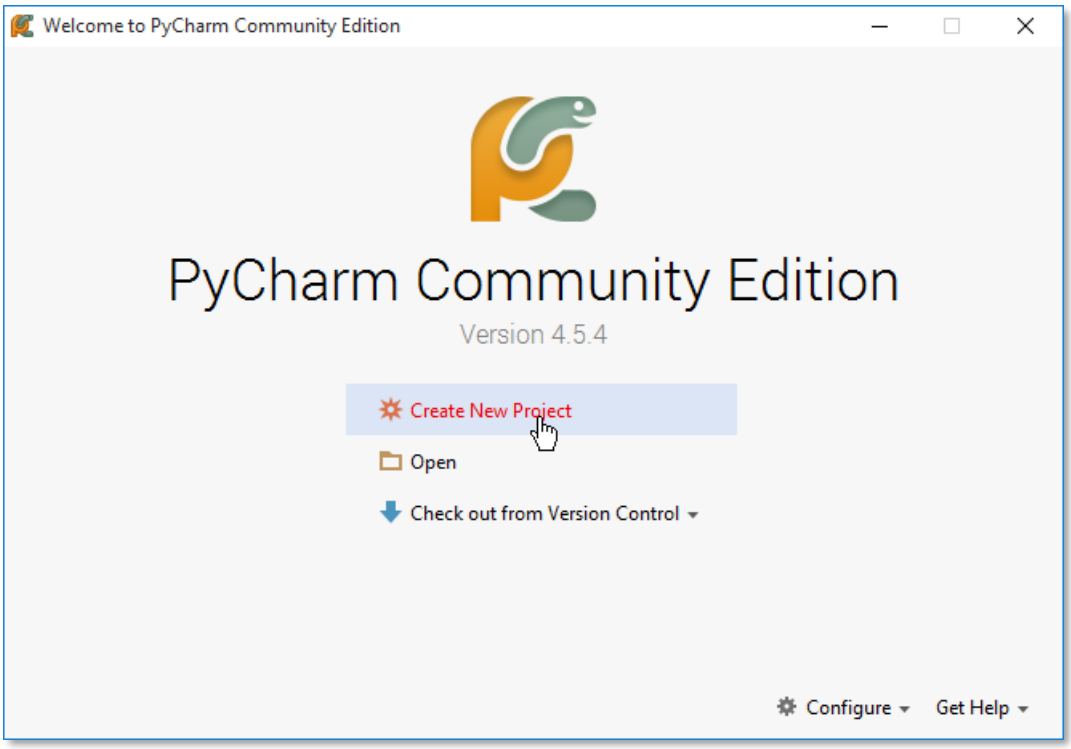


Рис. 6. Создаём новый проект

Если вы планируете писать собственные проекты, то для них лучше выбрать другую папку (см. текстовое поле *Location*), но мы пока проходим мимо (Рис. 7).

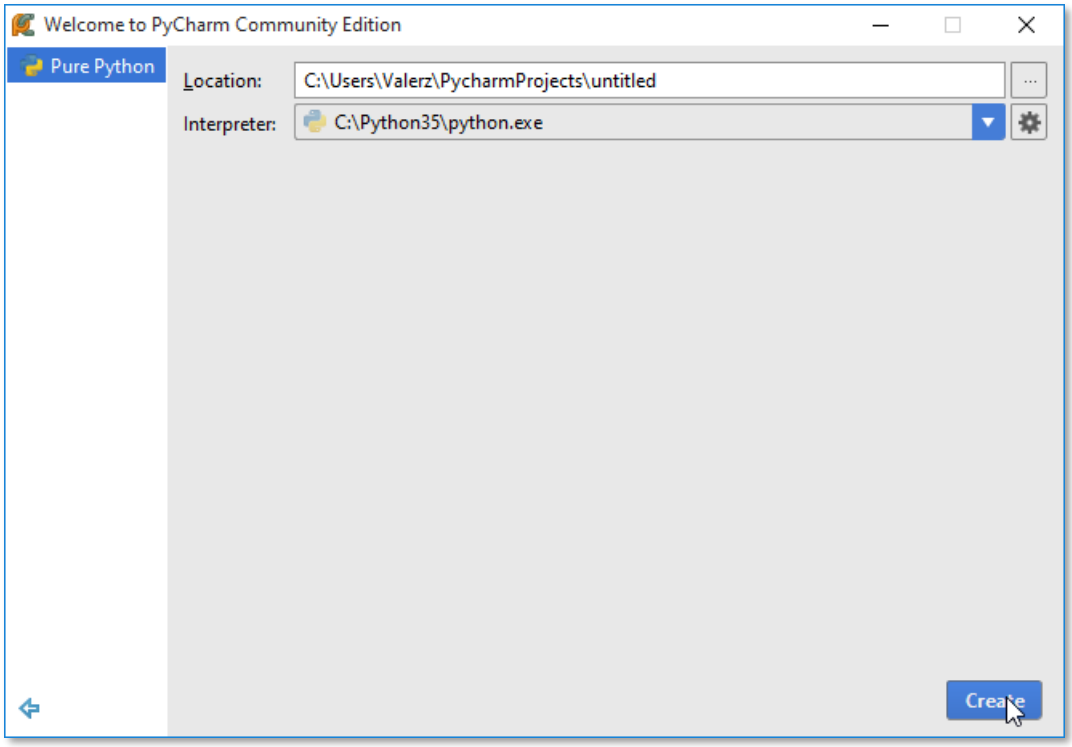


Рис. 7. Остаёмся в стандартной папке

Всё готово для яркой, плодотворной работы (Рис. 8).

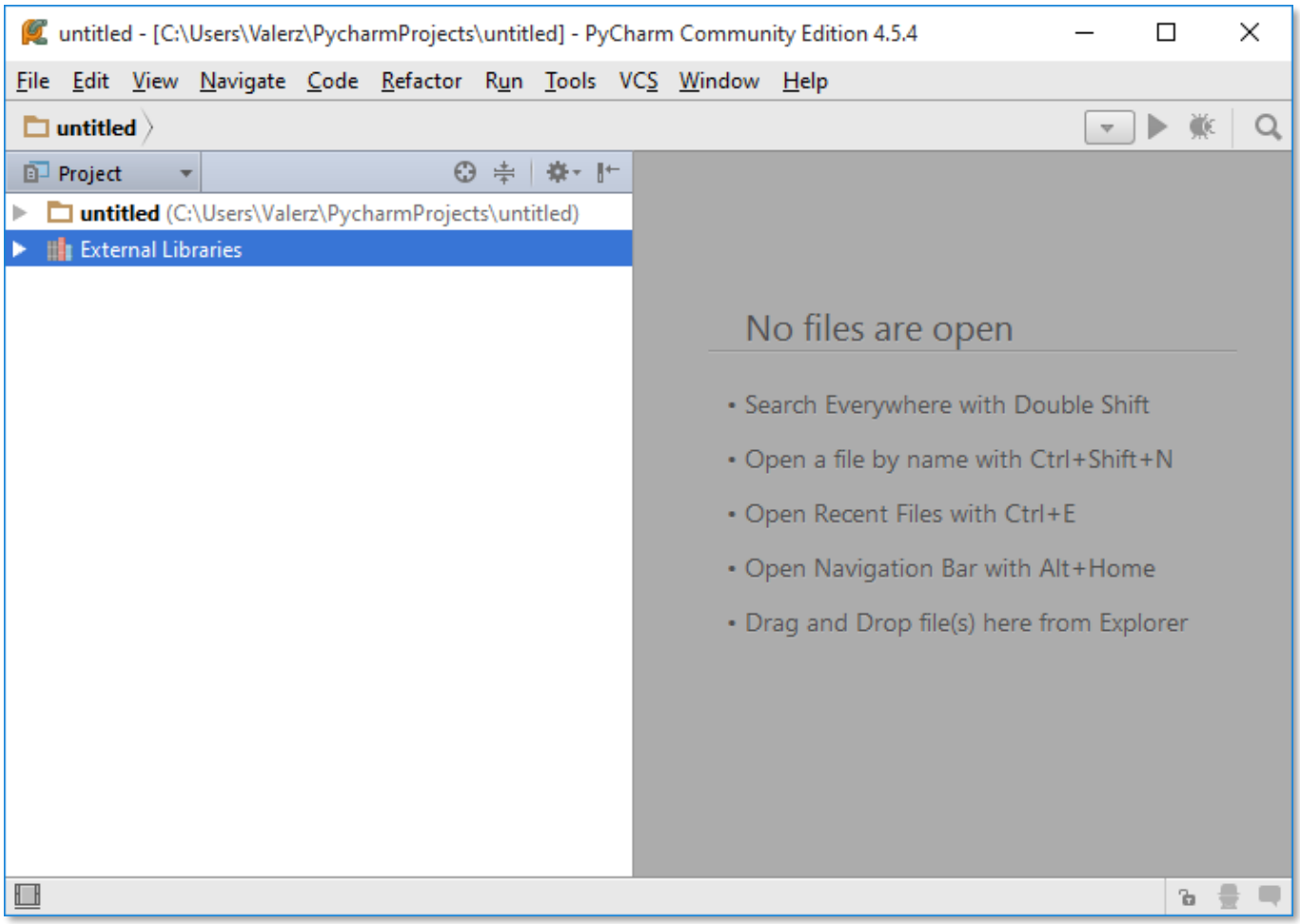


Рис. 8. Среда разработки готова!

Совершив ознакомительную экскурсию по среде разработки, нежно закройте её.

Устанавливаем *Задачник Абрамяна*

Поскольку мы преследуем вполне конкретную цель – решать задачи Абрамяна, то, прежде всего, нужно скачать и установить *Задачник* его имени.

На сайте http://ptaskbook.com/ru/ щёлкните по пункту меню **Скачивание дистрибутивов** (Рис. 1).

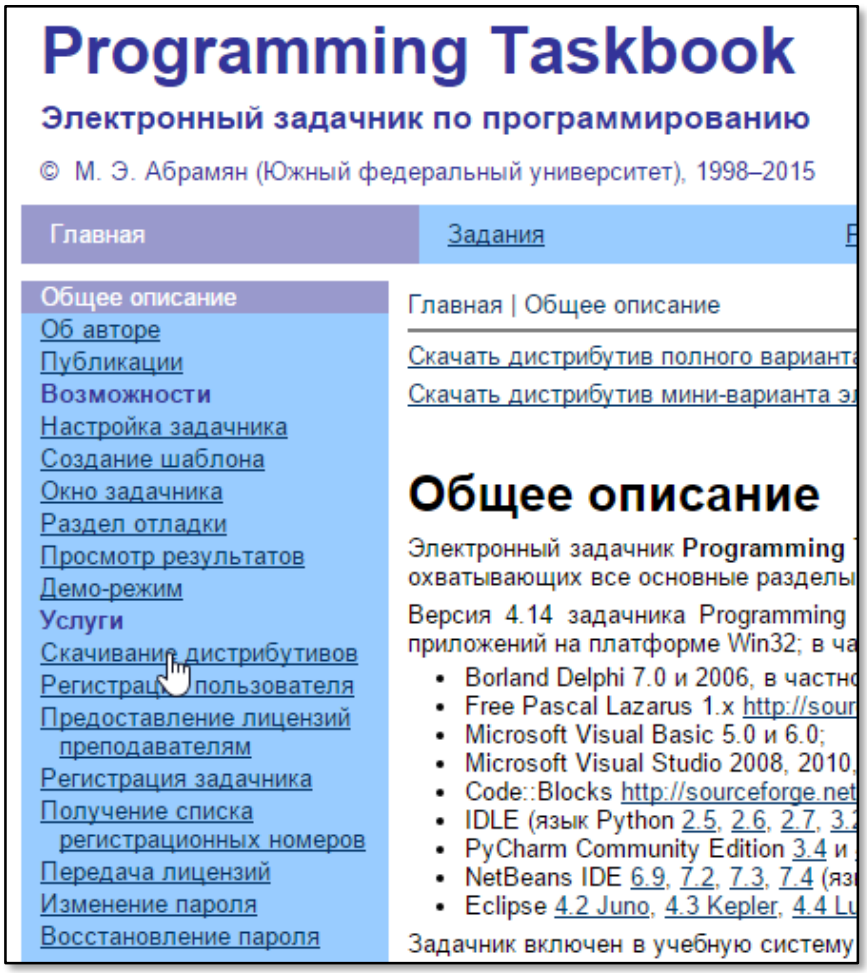


Рис. 1. Озадачиваемся

А затем – по указанной ниже строчке (Рис. 2).

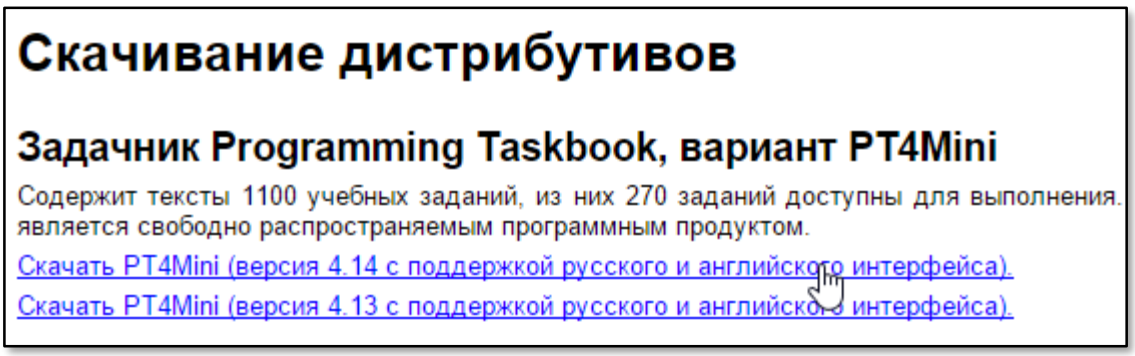


Рис. 2. Мини-задачник

Приготовьте папку и разархивируйте туда дистрибутив.

Запустите установочный файл **PT4Mini-ruen.exe** от имени *Администратора*.

Предпочтите русский язык интерфейса (Рис. 3).

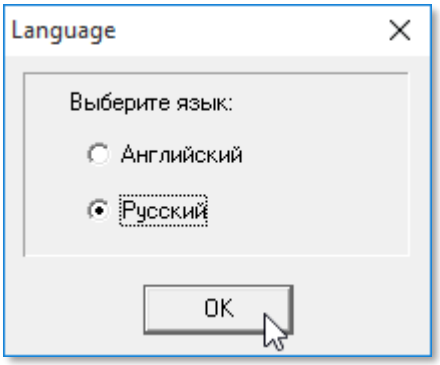


Рис. 3. Хорошо!

Папку для установки *Задачника* лучше выбрать на съёмном диске, если вы планируете работать с ним на разных компьютерах (Рис. 4).

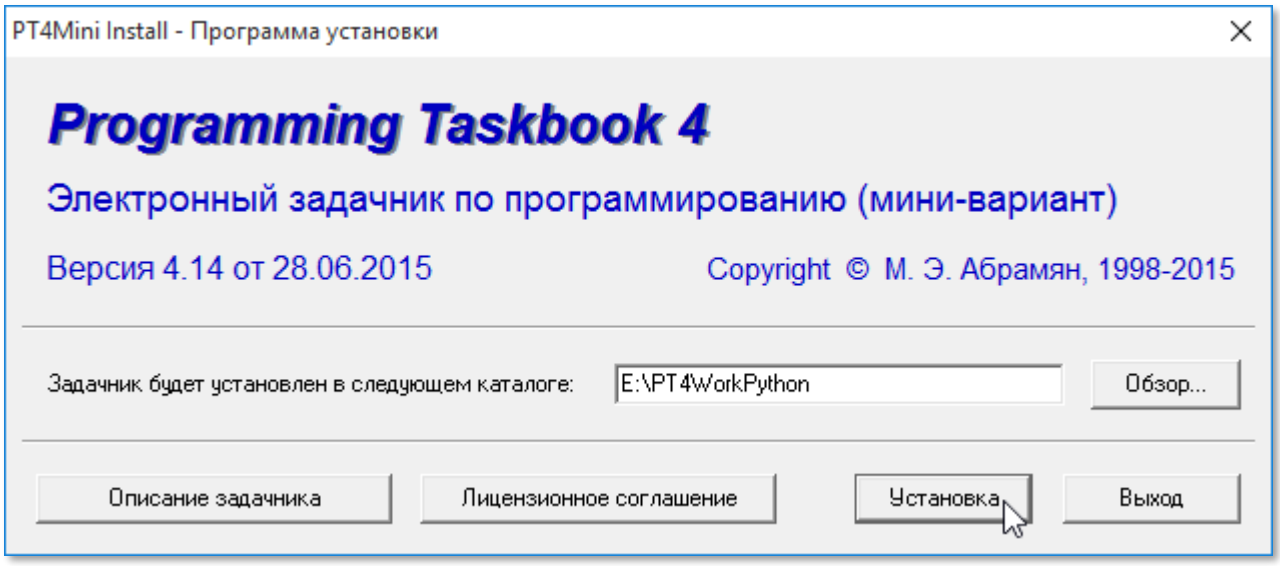


Рис. 4. Выбираем папку

Снимите лишние галочки и нажмите кнопку **ОК** (Рис. 5).

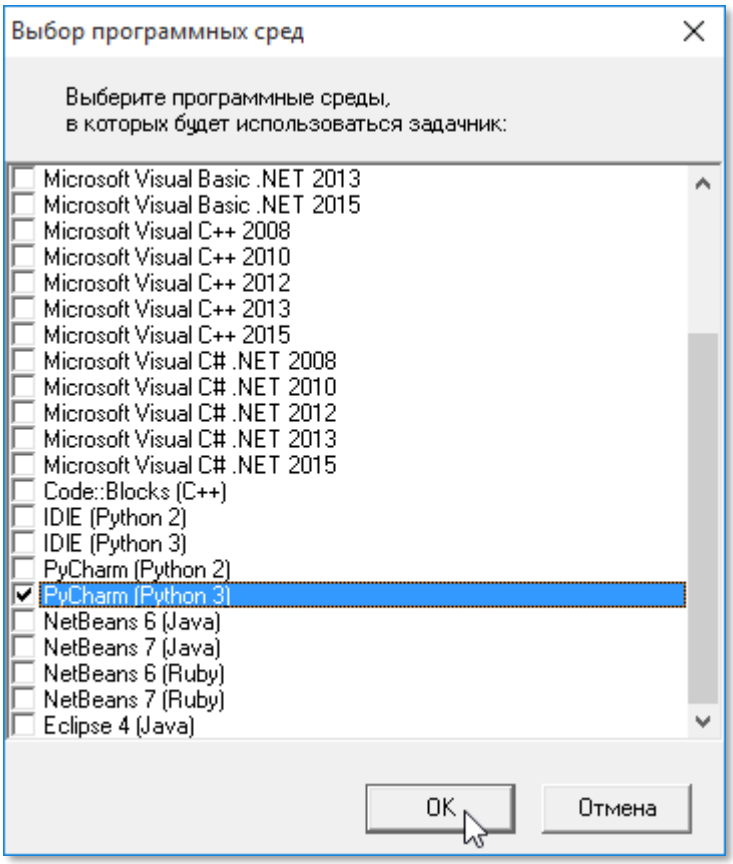


Рис. 5. Этих птичек нужно убрать

Пройдёт совсем немного времени, и вы получите радостное известие (Рис. 6).

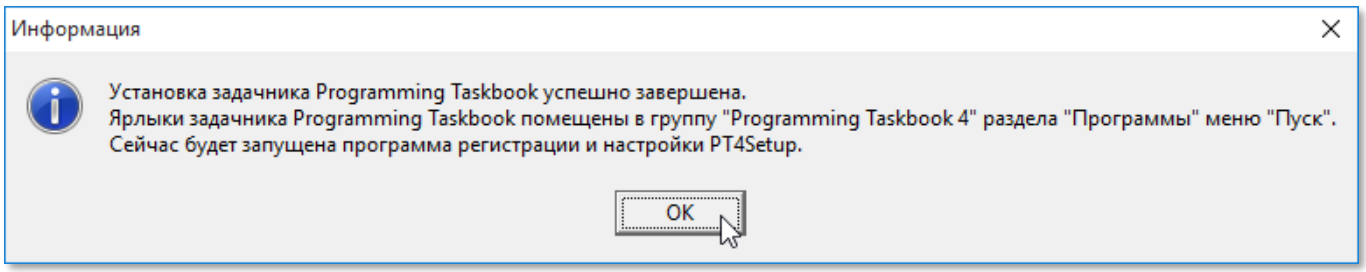


Рис. 6. Задачки в папке

Затем вам нужно **зарегистрироваться**, чтобы избежать анонимности при решении задач (Рис. 7).

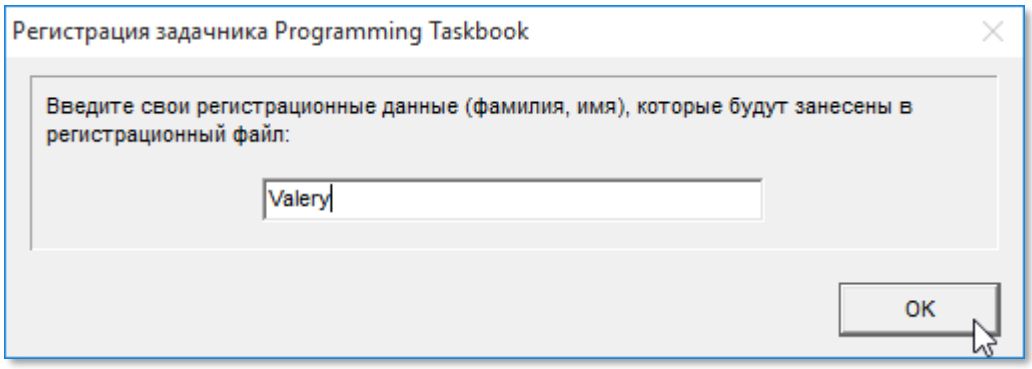


Рис. 7. Анонимность не допускается

Настроить *Задачник* на нужную среду разработки (Рис. 8).

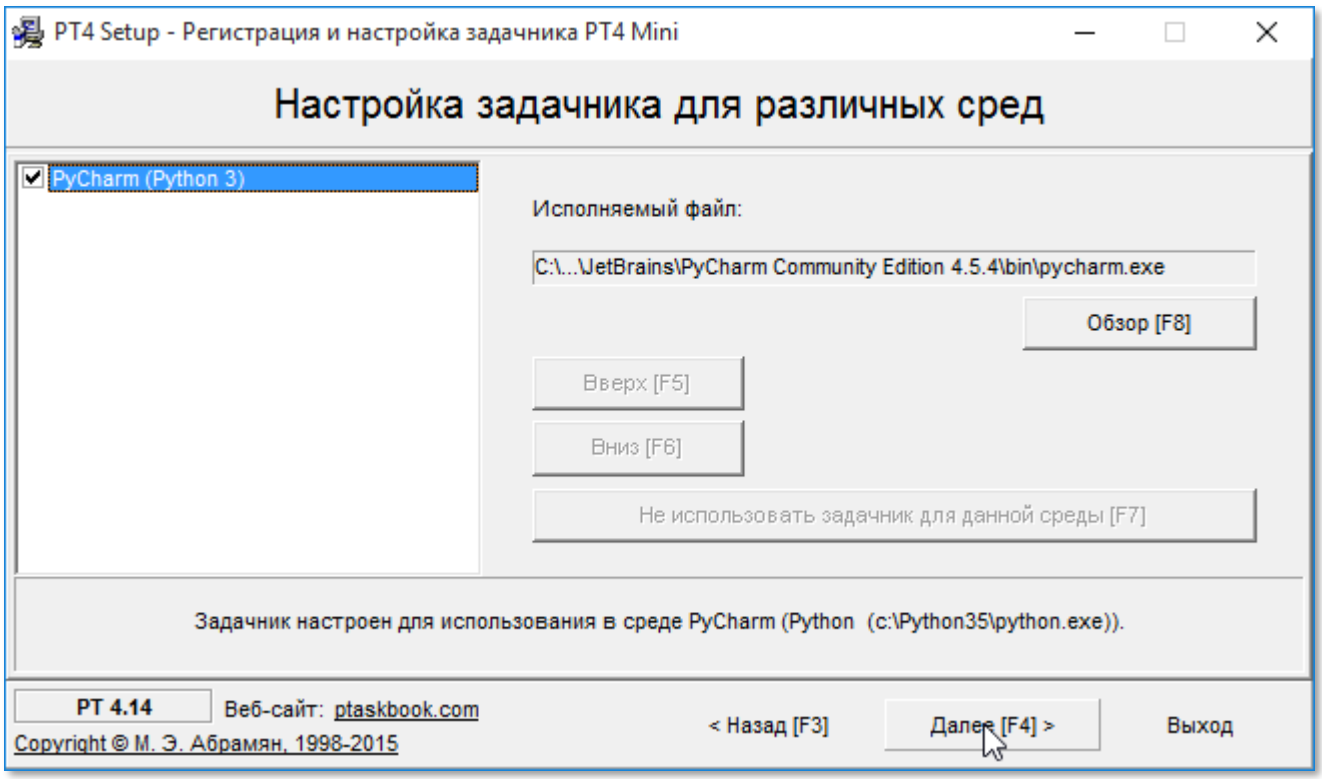


Рис. 8. Само собой

И в последнем диалоговом окне укажите рабочий каталог и создайте файл результатов, нажав соответствующую кнопку (Рис. 9).

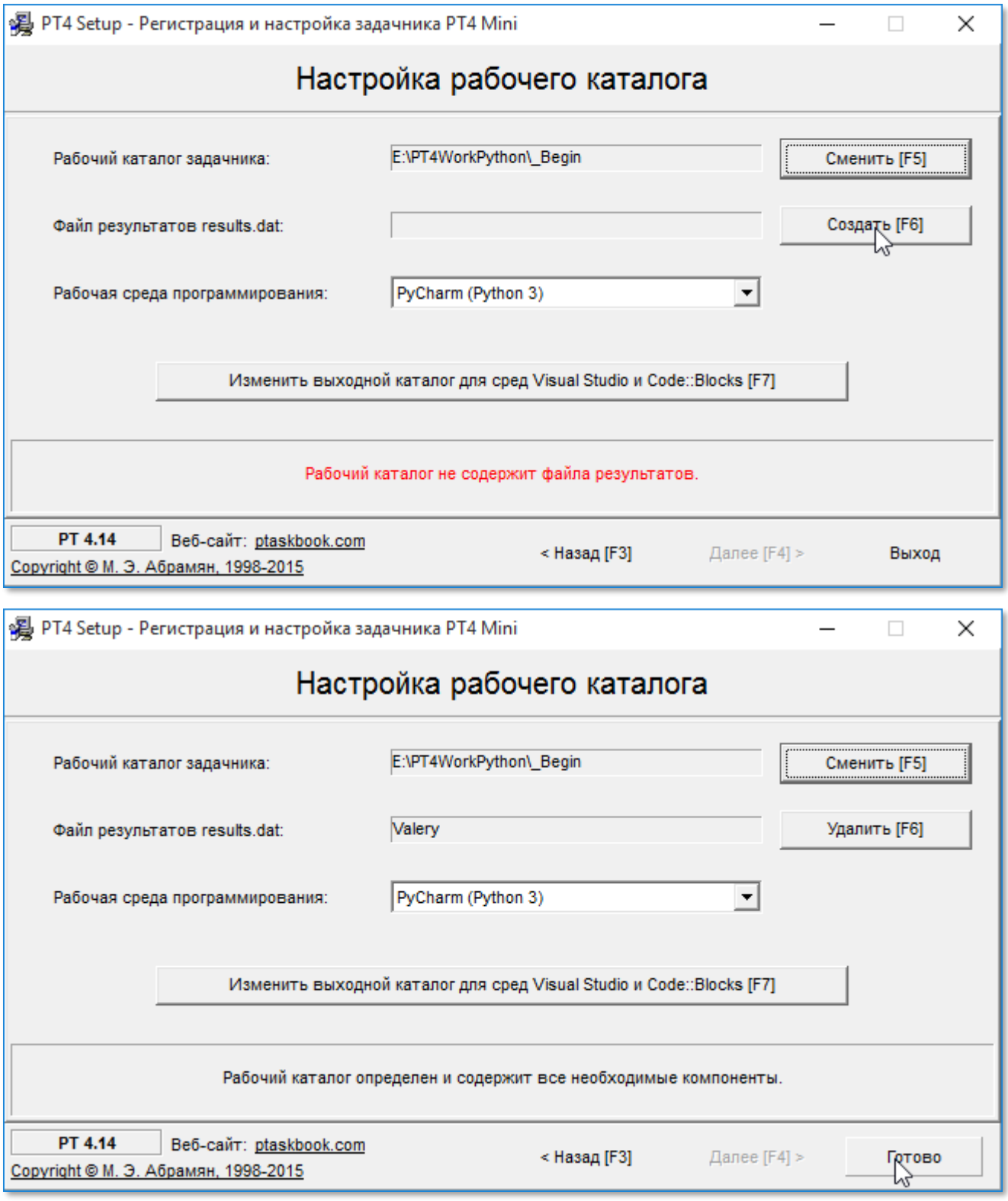


Рис. 9. Регистрация закончена

На *Рабочем столе* появятся новые **ярлыки** (Рис. 10).

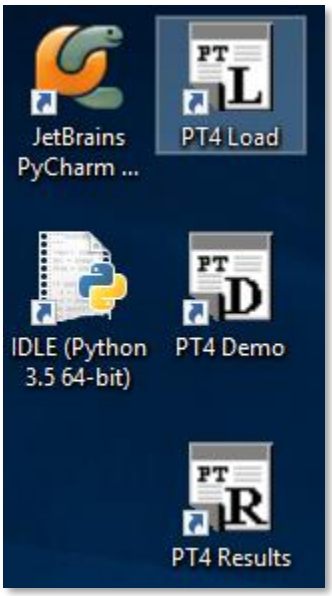


Рис. 10. Наклеиваем ярлыки

Электронный задачник

Пора **получить задание** для последующего его осмысливания и выполнения.

Щёлкните по ярлыку **PT4Load** на *Рабочем столе* (Рис. 1).

Рис. 1. Загружаем задачник

Откроется диалоговое окно *PT4 Load* (Рис. 2).

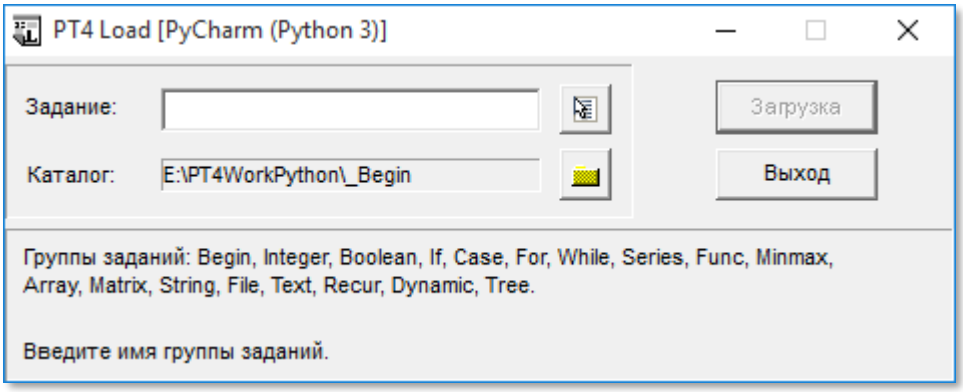


Рис. 2. Вступаем в диалог

Внизу вы можете прочитать, какие имеются группы заданий. Нам нужна группа **Begin**.

Набираем в текстовом поле *Задание* буквы *Be* (в любом регистре) и получаем внизу диалогового окна полное название группы (Рис. 3).

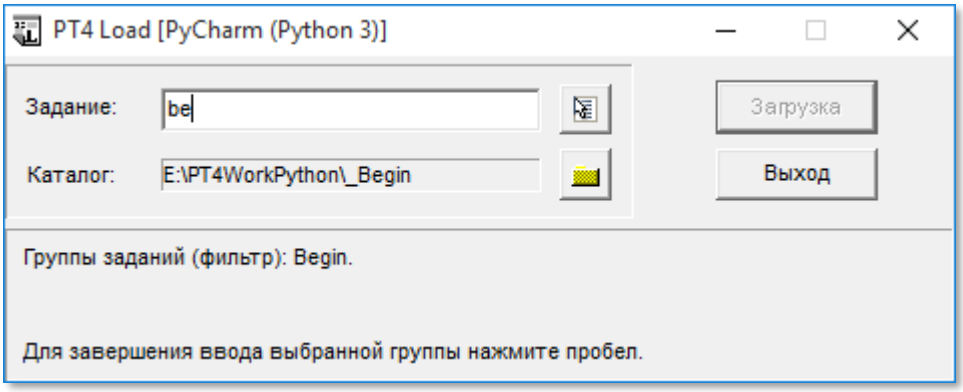


Рис. 3. Фильтр в действии

Достаточно нажать клавишу *ПРОБЕЛ*, чтобы название группы было закончено автоматически. Также вы получите подсказку: *в наборе 40 заданий* (Рис. 4).

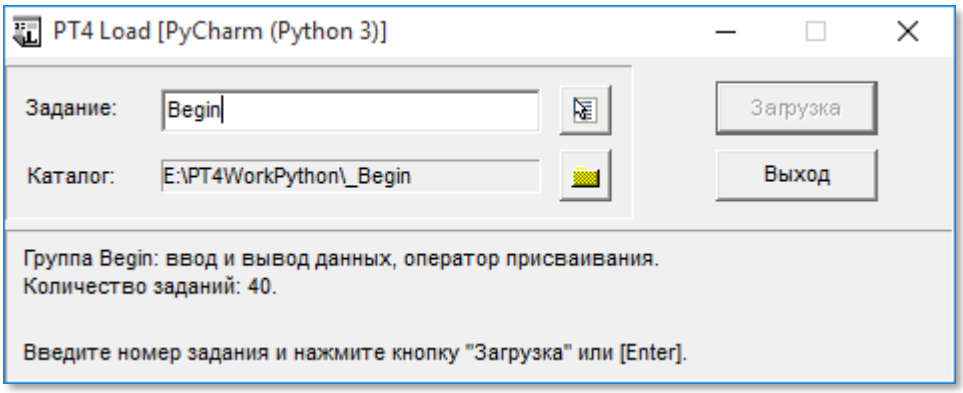


Рис. 4. Грамотно пользуйтесь подсказками

Конечно, начинать нужно с первого задания, поэтому печатаем цифру **1** (Рис. 5).

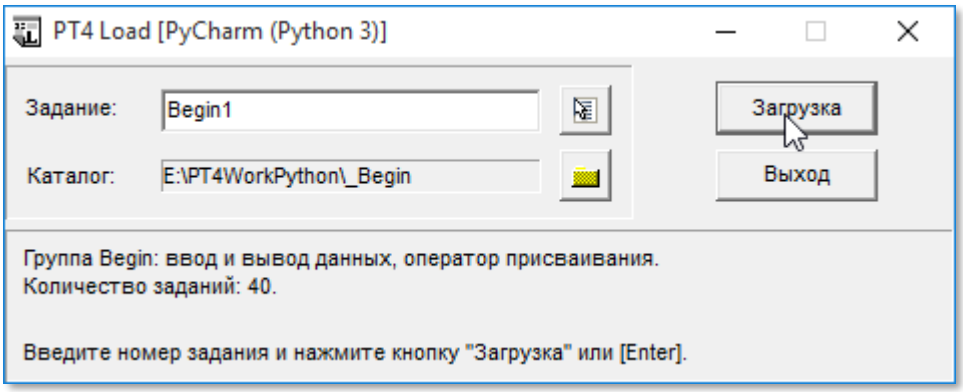


Рис. 5. Можно начинать

По умолчанию все ваши проекты-решения будут сохраняться в указанной при установке папке, название которой напечатано в текстовом поле *Каталог*. Если вы хотите изменить место хранения своих проектов, то нажмите кнопку с жёлтой папкой и укажите новый путь (Рис. 6).

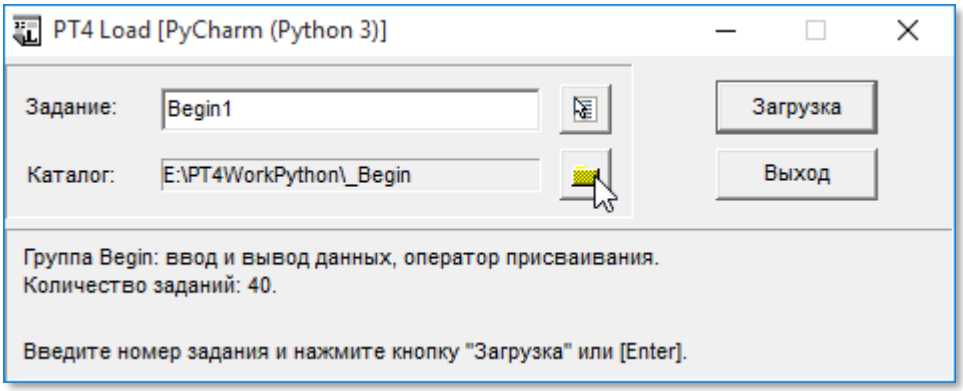


Рис. 6. Пути выбирают

Все приготовления закончены, и мы нажимаем кнопку *Загрузка* или клавишу *ВВОД* (Рис. 7).

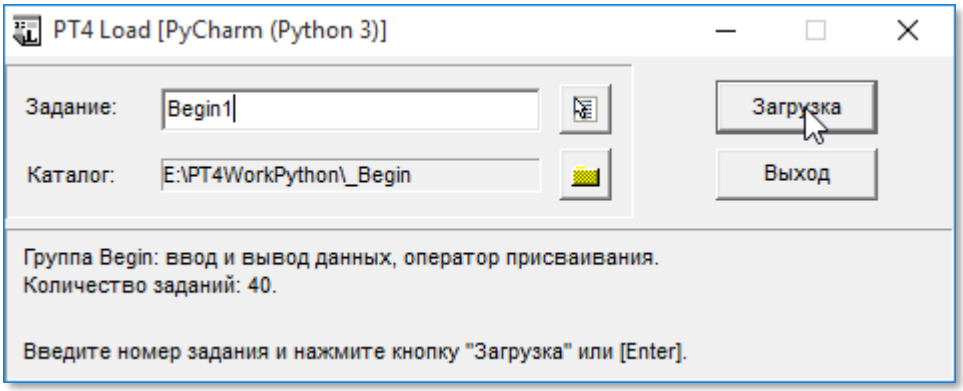


Рис. 7. Всё готово

Автоматически запустится программа *PyCharm* и в *Редакторе кода* появится страница *Begin1.pyw* с заготовкой кода для задания *Begin1* (Рис. 8).

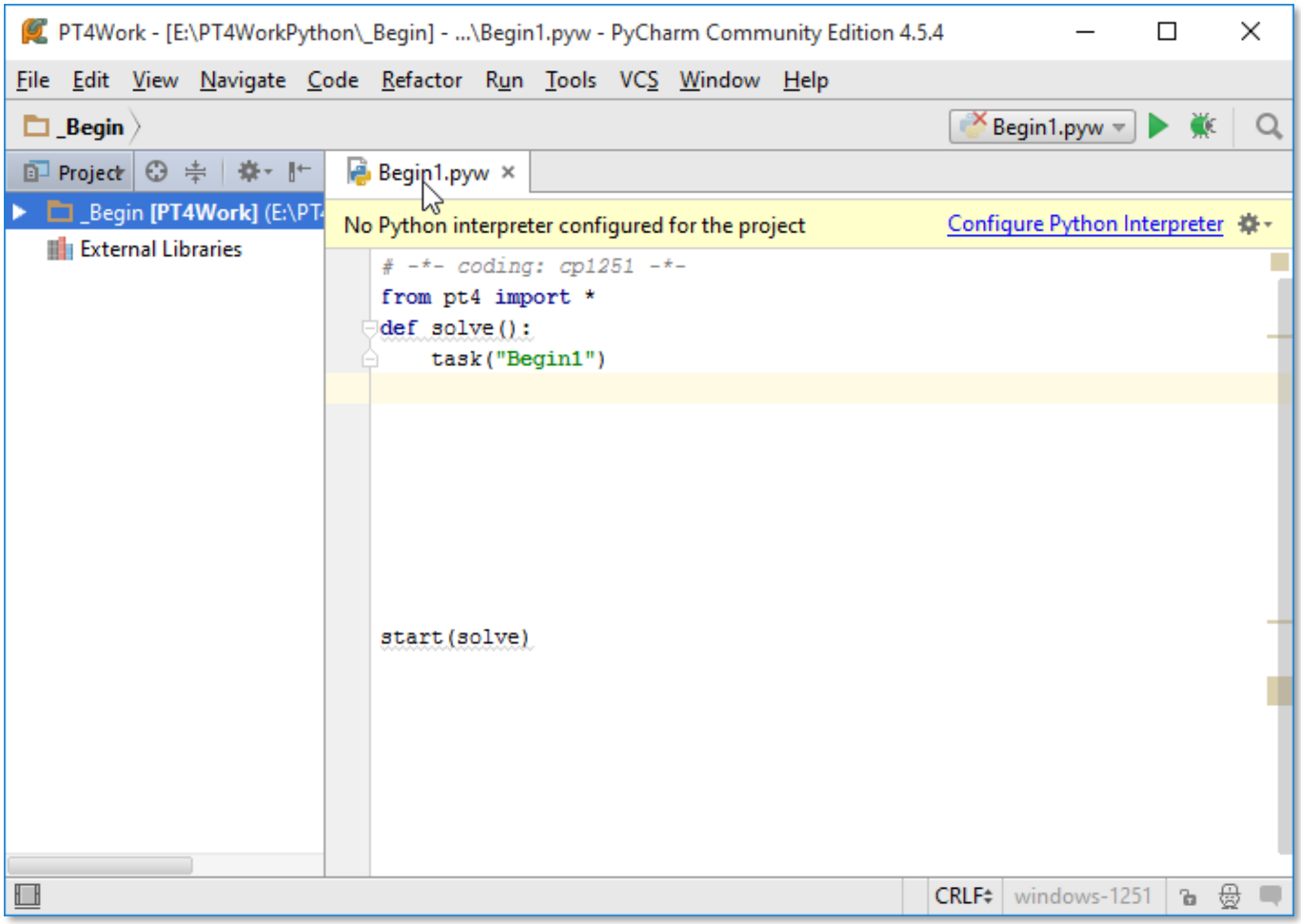


Рис. 8. Заготовительно-приготовительные работы закончены

Строчка

```
from pt4 import *
```

подключает к проекту модуль *pt4.py*, в котором находится *Электронный задачник Programming TaslBook 4*.

В функции **solve** записан вызов функции (процедуры) **task**, которой передаётся строка `"Begin1"` с названием задания.

> Для строковых литералов можно использовать и двойные кавычки, и одинарные.

Так как условие задачи пока неизвестно, то вы должны **запустить программу**. Обратите внимание на строку *Configure Python Interpreter* в правом верхнем углу окна *Редактора кода* (см. Рис. 8). Она сообщает, что перед запуском программы вы должны выбрать интерпретатор кода. Щёлкните по этой строке, чтобы открыть диалоговое окно **Settings**, выберите из списка установленную версию *Питона* и нажмите кнопку *OK* (Рис. 9).

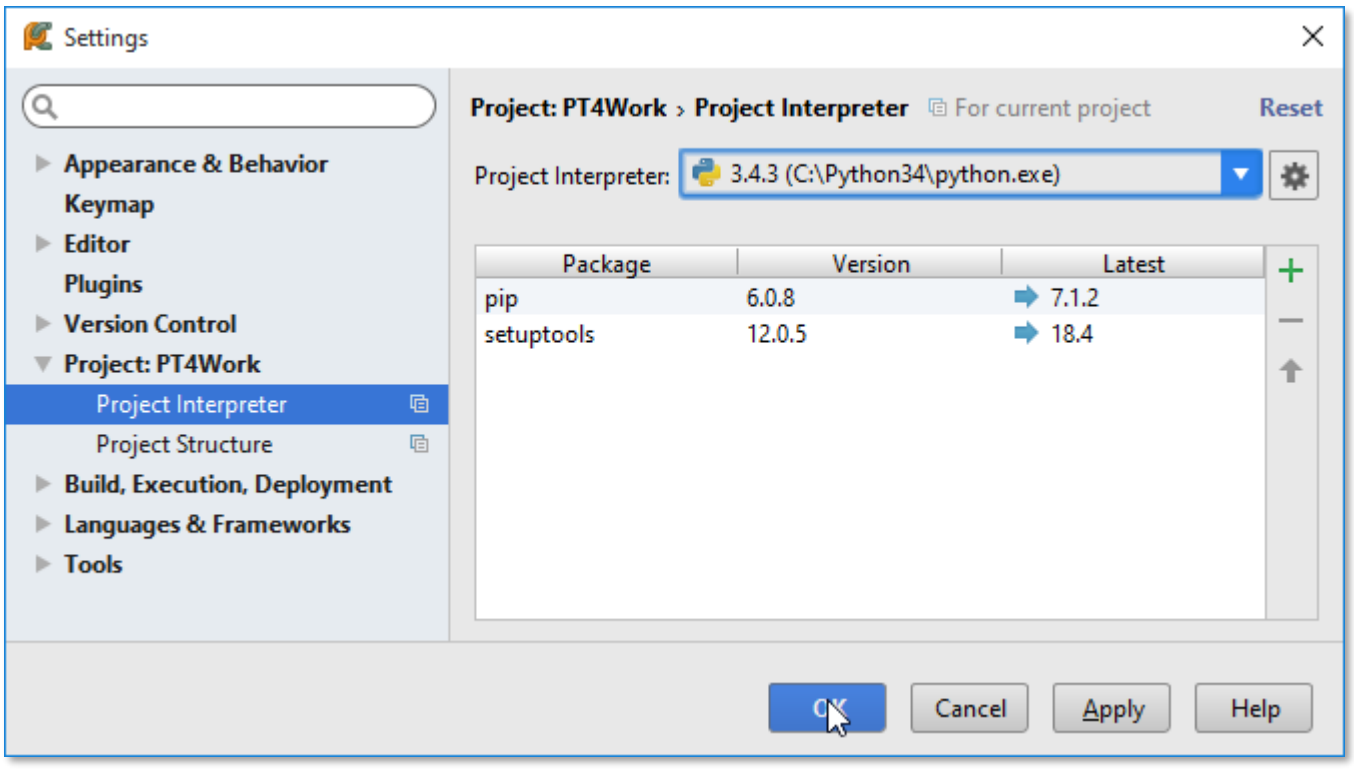


Рис. 9. Интерпретируй всё!

Теперь можно запустить программу (Рис. 10).

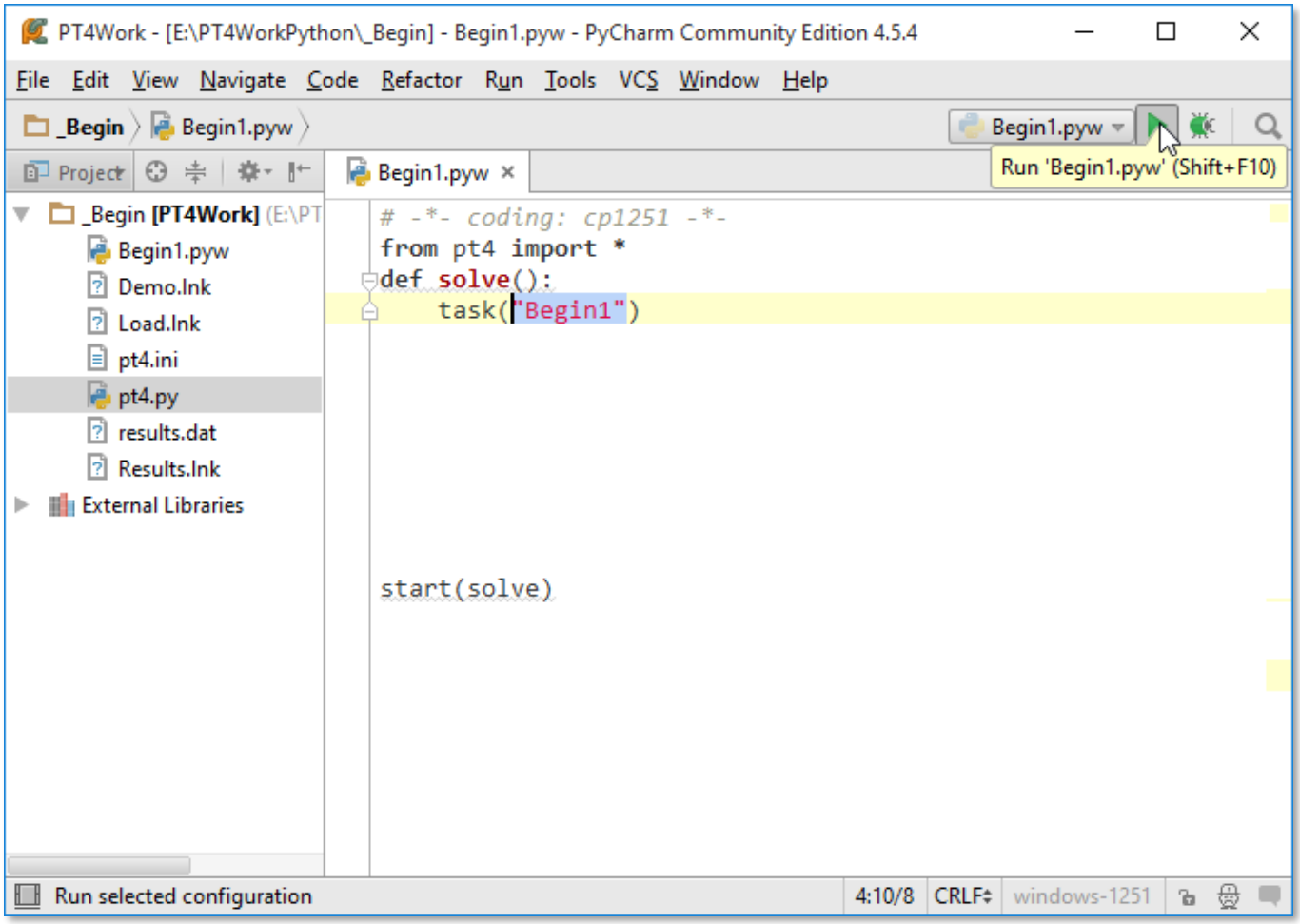


Рис. 10. Запускаем программу

На экране появится окно с заданием и элементами управления (Рис. 11).

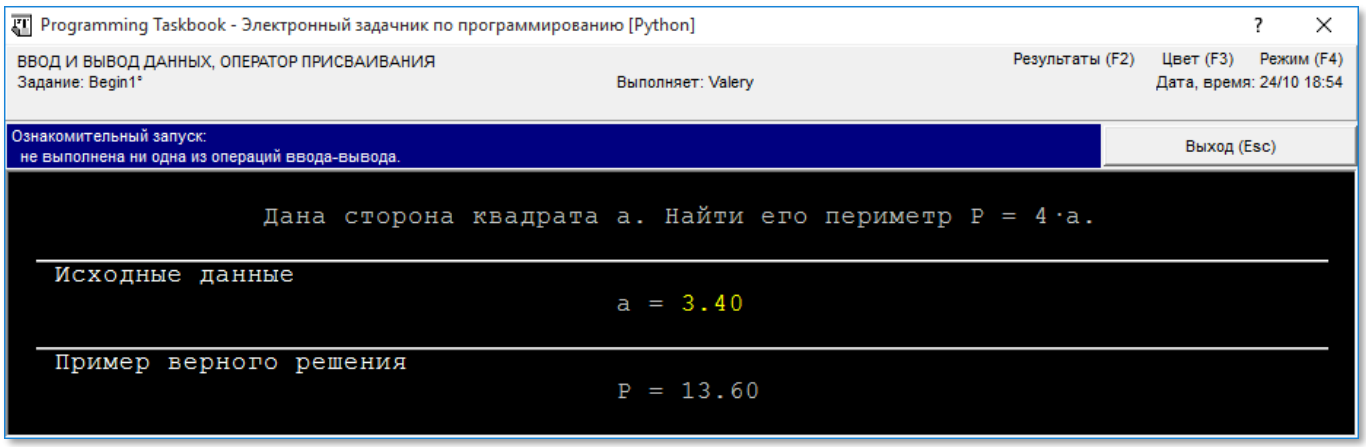


Рис. 11. Конкретное задание

Всегда внимательно читайте верхнюю строку (на чёрном фоне), в которой находится **условие задачи**. В средней строке вы можете прочитать исходные данные, а в нижней – правильный ответ. Исходные данные и ответ на задачу обновляются при каждом запуске программы, поэтому их нельзя использовать в исходном коде!

В **заголовке окна** напечатано название задачника, а также размещаются 2 кнопки. Кнопка с крестиком закрывает окно задания. Кнопка с вопросительным знаком открывает диалоговое окно **Информация** (Рис. 12), в котором вы найдёте исчерпывающие сведения по работе с задачником.

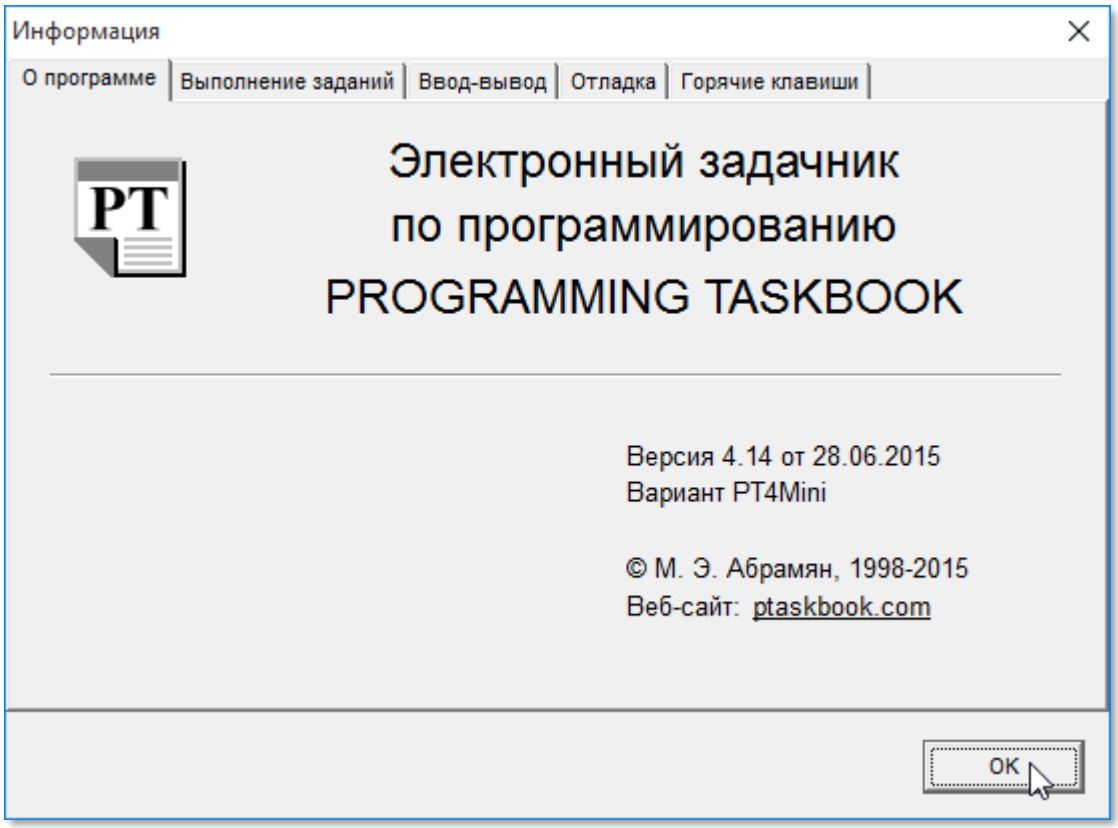


Рис. 12. Всегда держите под рукой!

Под заголовком вы можете прочитать название текущего задания «*Begin1*», имя «испытуемого», а также содержание первой группы заданий: *ВВОД И ВЫВОД ДАННЫХ, ОПЕРАТОР ПРИСВАИВАНИЯ*.

Правее находятся **кнопки**:

- Кнопка **Результаты** (клавиша *F2*) вызывает окно, показывающее, какие задания и насколько успешно вы уже решали (Рис. 13).

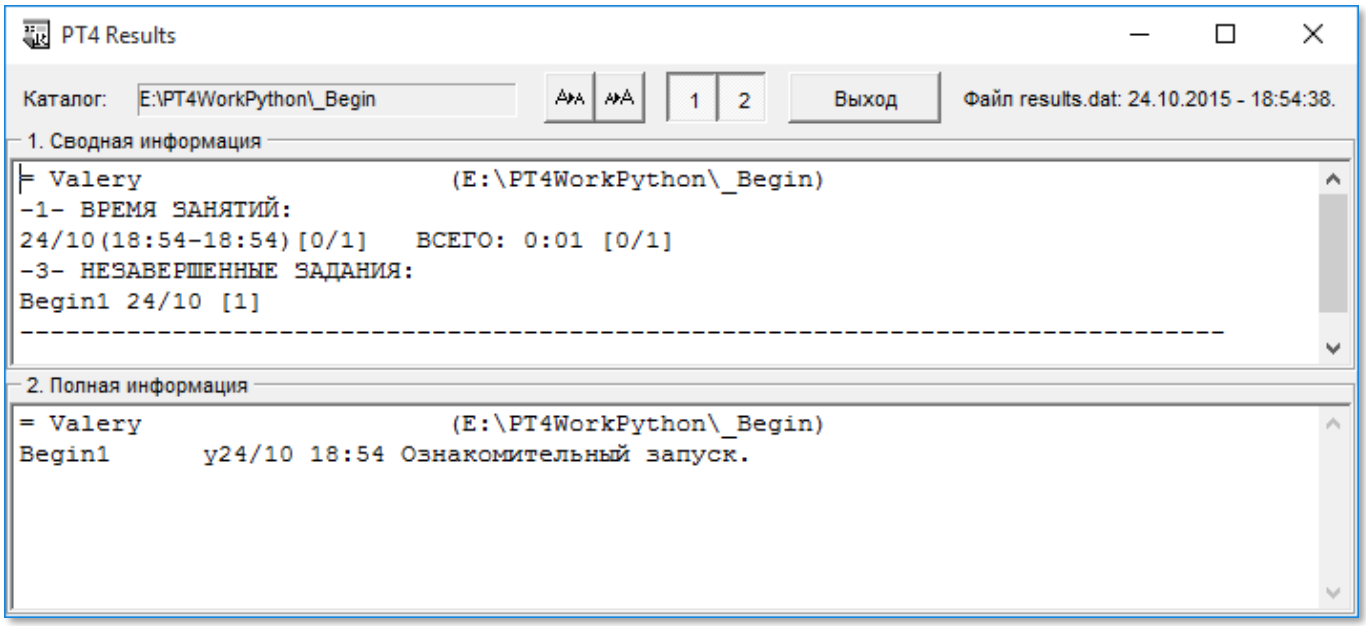


Рис. 13. Контора пишет!

- Кнопка **Цвет** (клавиша *F3*) инвертирует цвета фона и шрифта в строках с заданием (Рис. 14).

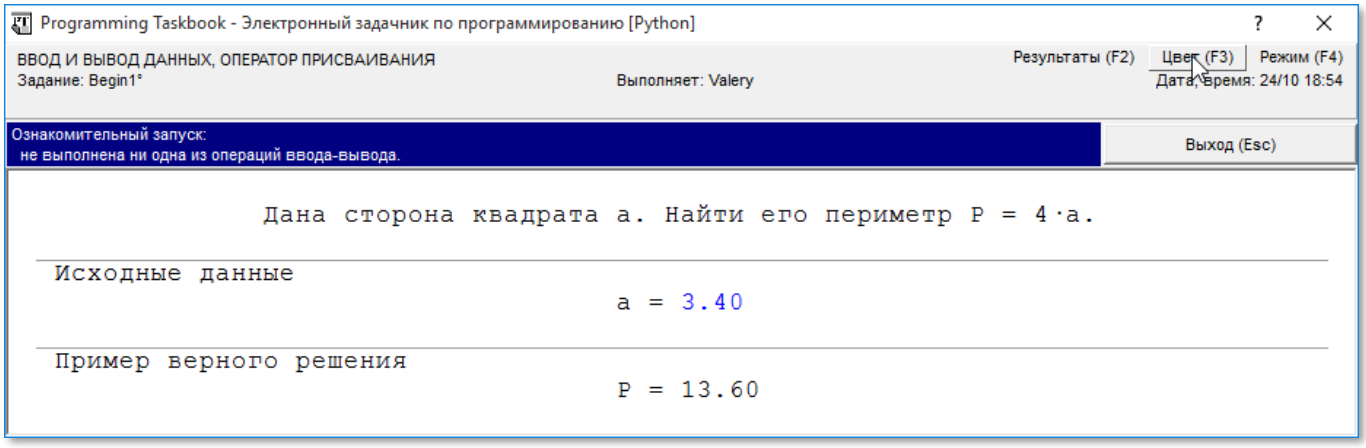


Рис. 14. Позитив негатива

Если вам больше по душе писать чёрным по белому, чем наоборот, то вы можете остановиться на этом колористическом варианте. Повторное нажатие на кнопку *Цвет* вернёт этим строчкам первоначальный мрачный вид.

- Кнопка **Режим** изменяет формат окна задания (Рис. 15).

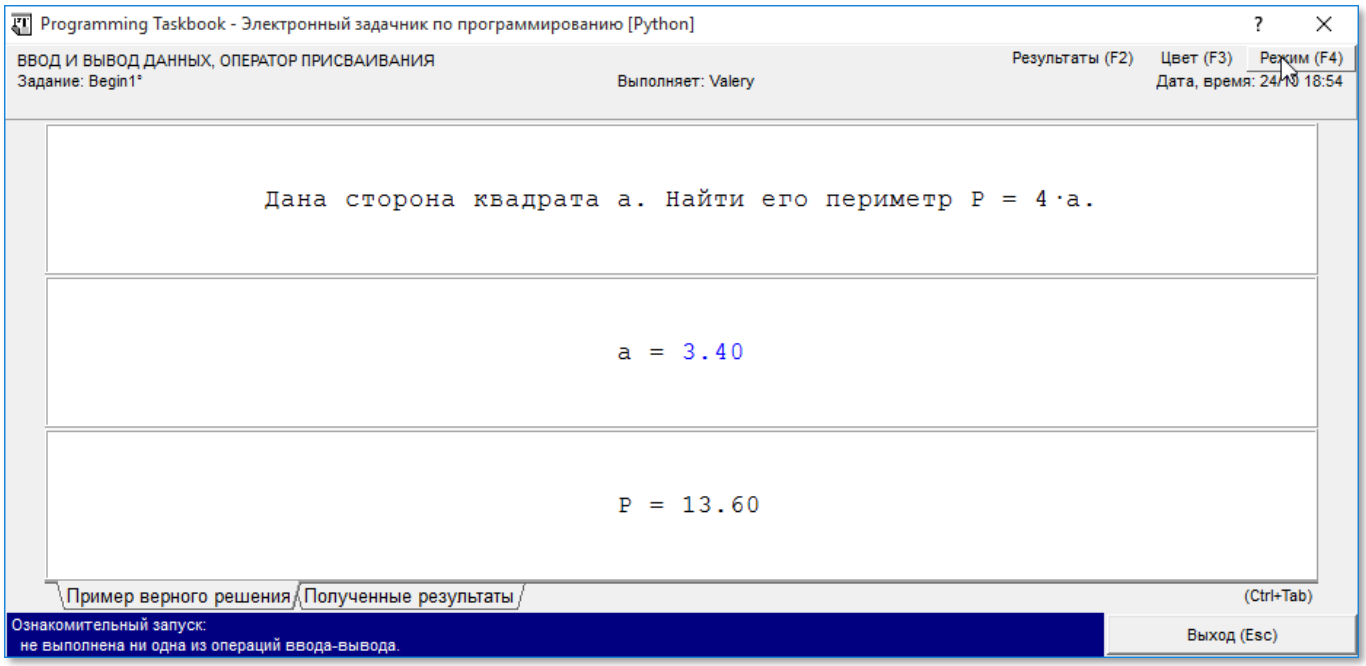


Рис. 15. Не формат?

Повторное нажатие на эту кнопку возвращает окно в исходное состояние.

И наконец, под этими кнопками выводится текущая дата и время.

Обратите внимание, что задачник очень умный и понимает, что мы пока не решаем задачу, а только знакомимся с её условием. Об этом говорит сообщение в синем текстовом поле: *Ознакомительный запуск: не выполнена ни одна из операций ввода-вывода.*

Запомните или запишите условие задания и закройте окно. Если память у вас слабая или лень переписывать задание в тетрадь, то окно можно и не закрывать. Тогда при необходимости вы всегда сможете по(д)смотреть условие задачи.

Итак, **первое задание** такое:

Дана сторона квадрата a. Найти его периметр P = 4·a.

Как вы видите, решать ничего не нужно, поскольку формула для вычисления периметра приведена в условии задания. Нам нужно только записать её на *Питоне*.

Возвращаемся в *Редактор кода* и после вызова функции *task* получаем от *Задачника* значение вещественного типа для переменной **a**:

```
#сторона квадрата:
#получаем значение переменной:
a = get_float()
```

Именно этой буквой длина стороны квадрата обозначена в формуле. Чтобы не ничего не перепутать, лучше не менять названия переменных.

Помните, что **во всех заданиях переменные имеют вещественный тип *float*.**

Обычный способ ввода значений в языке *Питон* – с помощью функции **input**, но при решении задач Абрамяна из набора *Begin* следует использовать функцию **get_float**, которая возвращает вещественное значение.

> Допускается и универсальная функция **get**, но её действие менее понятно в коде.

После выполнения нашего кода переменная *a* получит от *Задачника* некоторое *случайное* значение. Какое именно – нам знать необязательно, так как оно просто используется для вычисления периметра по формуле:

$$P = 4 \cdot a$$

Для **идентификаторов переменных в *Питоне* принято использовать строчные буквы**, поэтому переменная для хранения длины периметра должна называться *p*.

Обратимся к правой части уравнения для вычисления периметра. Её значение равно произведению значения переменной *a* на 4. Точку следует заменить звёздочкой - знаком умножения, принятом в программировании.

Четвёрка – это литерал целого типа, а переменная *a* имеет вещественный тип. В результате выполнения операции умножения результат получит более «высокий» тип, то есть вещественный тип переменной *a*.

Оператор для вычисления периметра готов:

```
#периметр:
p = 4 * a
```

И последняя обязательная операция во всех заданиях – **печать результата** выполнения задачи на экране. В *Питоне* за это отвечает функция *print*, но при решении задач Абрамяна нужно использовать функцию **put**:

```
#печатаем результат на экране:
put(p)
```

Нажимаем кнопку **Run** – и получаем от программы сообщение, что **с первым заданием мы успешно справились** (Рис. 16)!

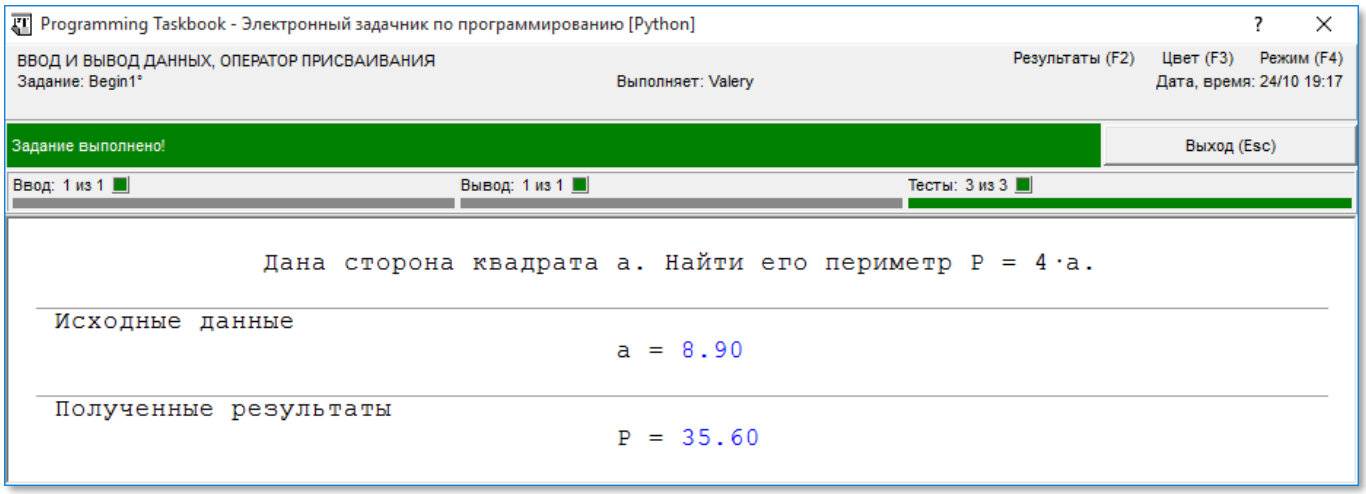


Рис. 16. С почином!

Итак, **полный код программы** должен быть таким:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin1")

    #сторона квадрата:
    #получаем значение переменной:
    a = get_float() #get()

    #периметр:
    p = 4 * a
    #печатаем результат на экране:
    put(p)


start(solve)
```

Во всех заданиях должны быть:

- *функции ввода и вывода get_float/get и put*
- *арифметические/математические выражения*
- *операторы присваивания*

Запомните:

- *Все переменные имеют вещественный тип float.*
- *Если числовой литерал имеет дробную часть, то она отделяется от целой части точкой, а не запятой.*
- *Знак операции присваивания – это обычный знак равенства.*
- *Идентификаторы переменных пишутся с маленькой буквы.*
- *Тип переменной определяется типом значения правой части в операторе присваивания.*
- *Оператор присваивания действует так. Сначала вычисляется значение правой части, а затем оно присваивается переменной в левой части.*
- *В выражениях всегда используется текущее значение переменной.*

Задачник строго следит за выполнение этих правил и в случае **нарушений** выдаёт соответствующие сообщения.

Давайте закомментируем оператор вывода:

```
#печатаем результат на экране:
#put(p)
```

И запустим программу.

В окне приложения появляется **оранжевая** строка, сообщающая нам, что мы напечатали **не все** необходимые результаты, а именно 0 вместо 1 (Рис. 17).

Если вы забудете объявить переменную:

```
#периметр:
#p = 4 * a
#печатаем результат на экране:
put(p)
```

то получите строгое предупреждение (Рис. 18).

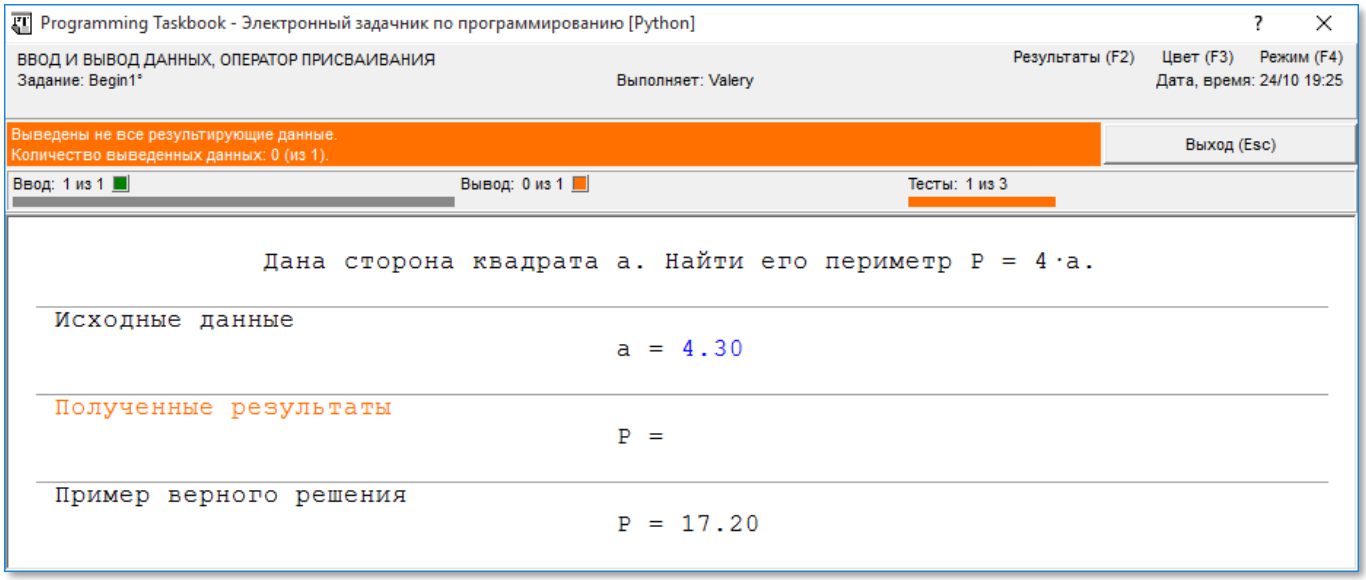


Рис. 17. Недостача!

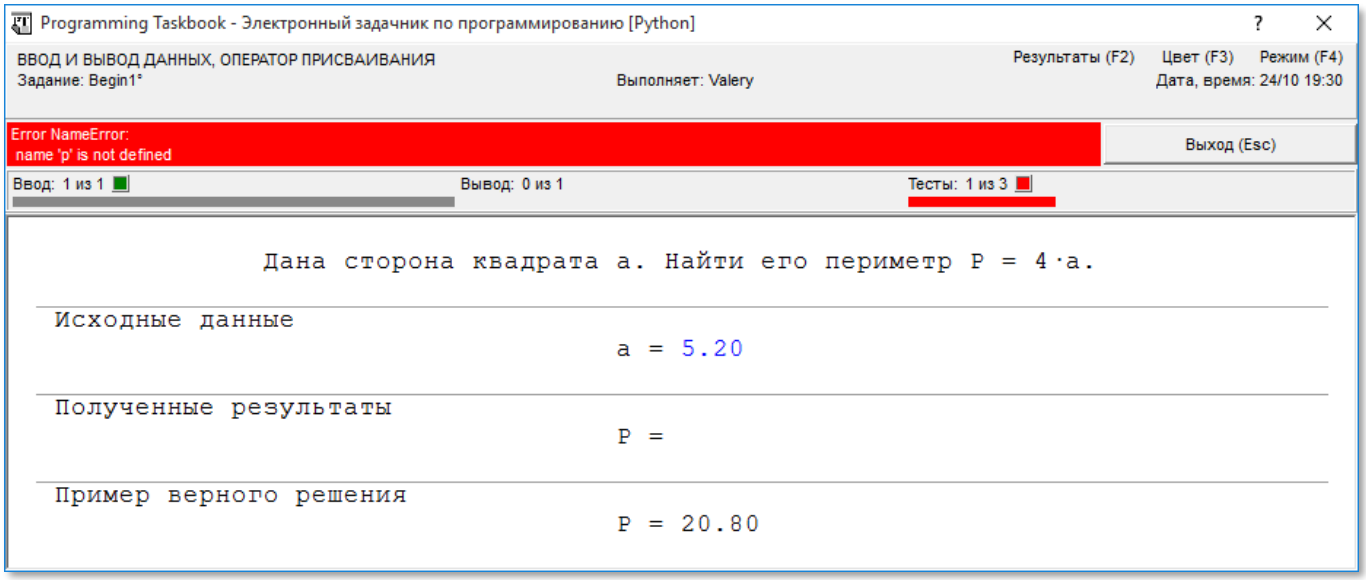


Рис. 18. Не забывай!

Действительно, в функции *put* указана переменная **p**, которая до этого не объявлена. Компилятор её не знает!

Аналогично вы может сделать и другие ошибки, чтобы посмотреть, как на них реагируют компилятор и *Задачник*. Впрочем, если вы будете невнимательны, то ошибки в программах будут возникать сами собой, без дополнительных усилий.

Дополнительные функции ввода

Электронный задачник имеет и другие функции для ввода значений. Вы также можете использовать их при решении задач (но не в других программах!):

get_bool() – возвращает значение логического типа
get_int() – возвращает значение целого типа
get_str() – возвращает значение строкового типа

Задание Begin2

Получить заготовку для второго задания ещё проще, чем для первого!

Запустите программу *PT4 Load* и исправьте в диалоговом окне единицу на двойку (Рис. 1).

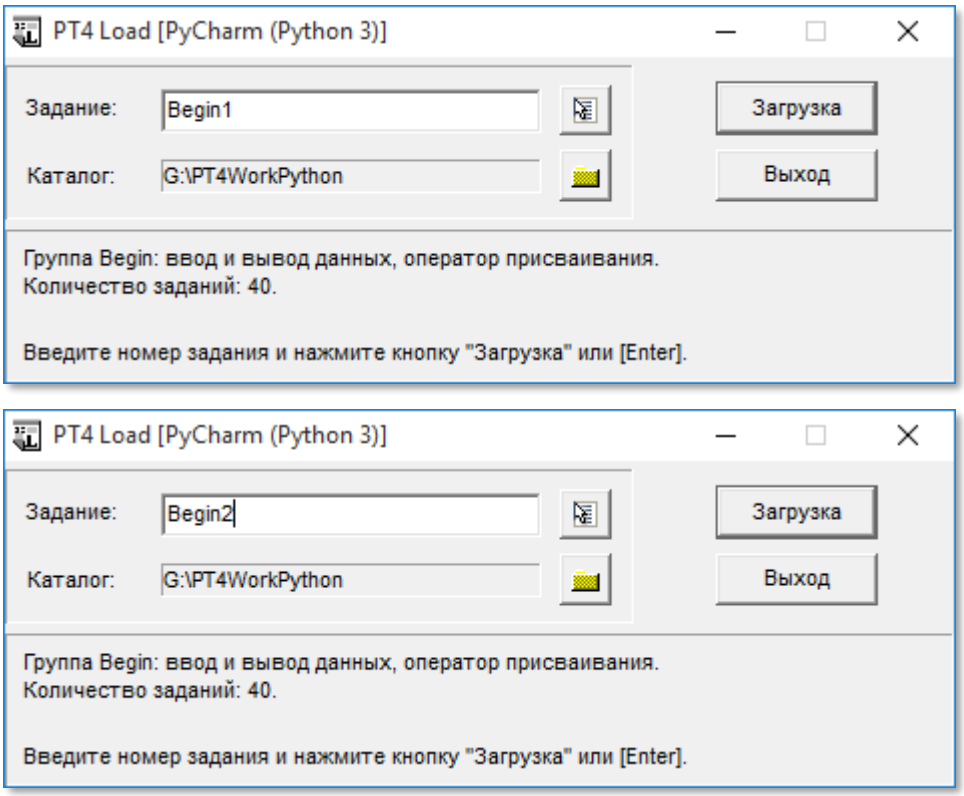


Рис. 1. Идём дальше

В *Редакторе кода* откроется **заготовка**:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin2")


start(solve)
```

Так как прежде чем браться за выполнение задания, его нужно увидеть, то запустите программу и внимательно прочитайте задание.

Другой способ ознакомиться с заданием – запустить программу **PT4Demo**.

В открывшемся диалоговом окне (Рис. 2) выберите из списка нужную группу заданий, номер задания и нажмите кнопку **Просмотр**.

Откроется диалоговое окно с условием задачи (Рис. 3).

Если вы хотите разом увидеть условия всех заданий и/или распечатать их, то нажмите кнопку *Показать группу заданий в виде html-страницы* (клавиатурное сокращение *F2*) (Рис. 4).

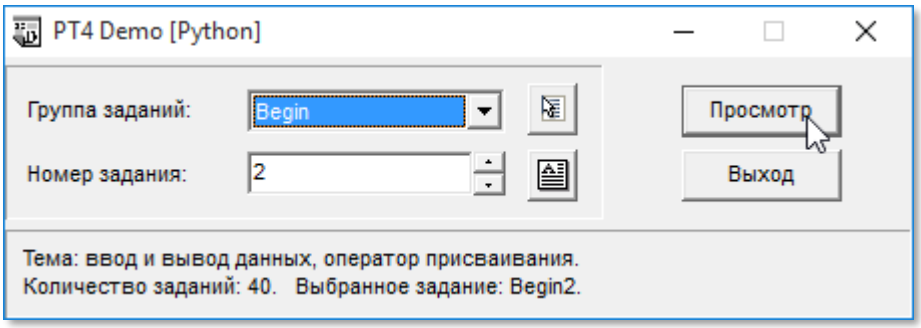


Рис. 2. Кнопочная демонстрация

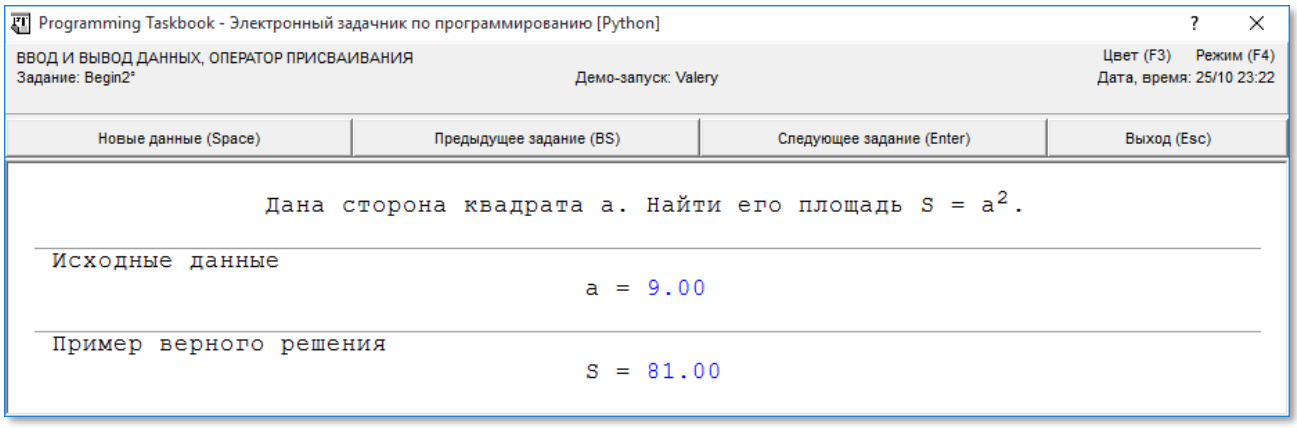


Рис. 3. Отличный обзор!

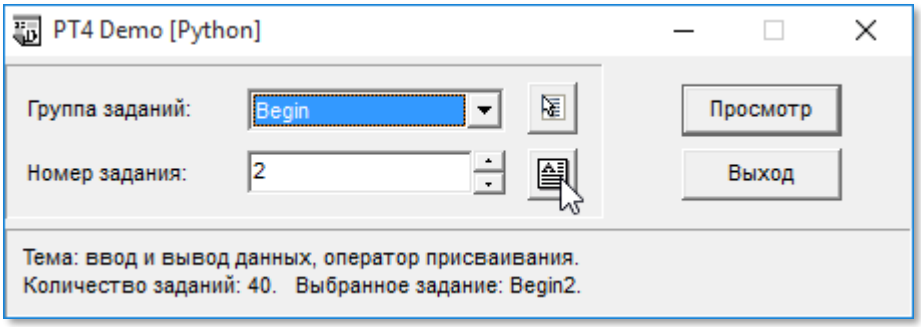


Рис. 4. Другая кнопка

В браузере откроется страница, и вы сможете прочитать условие любого задания (Рис. 5).

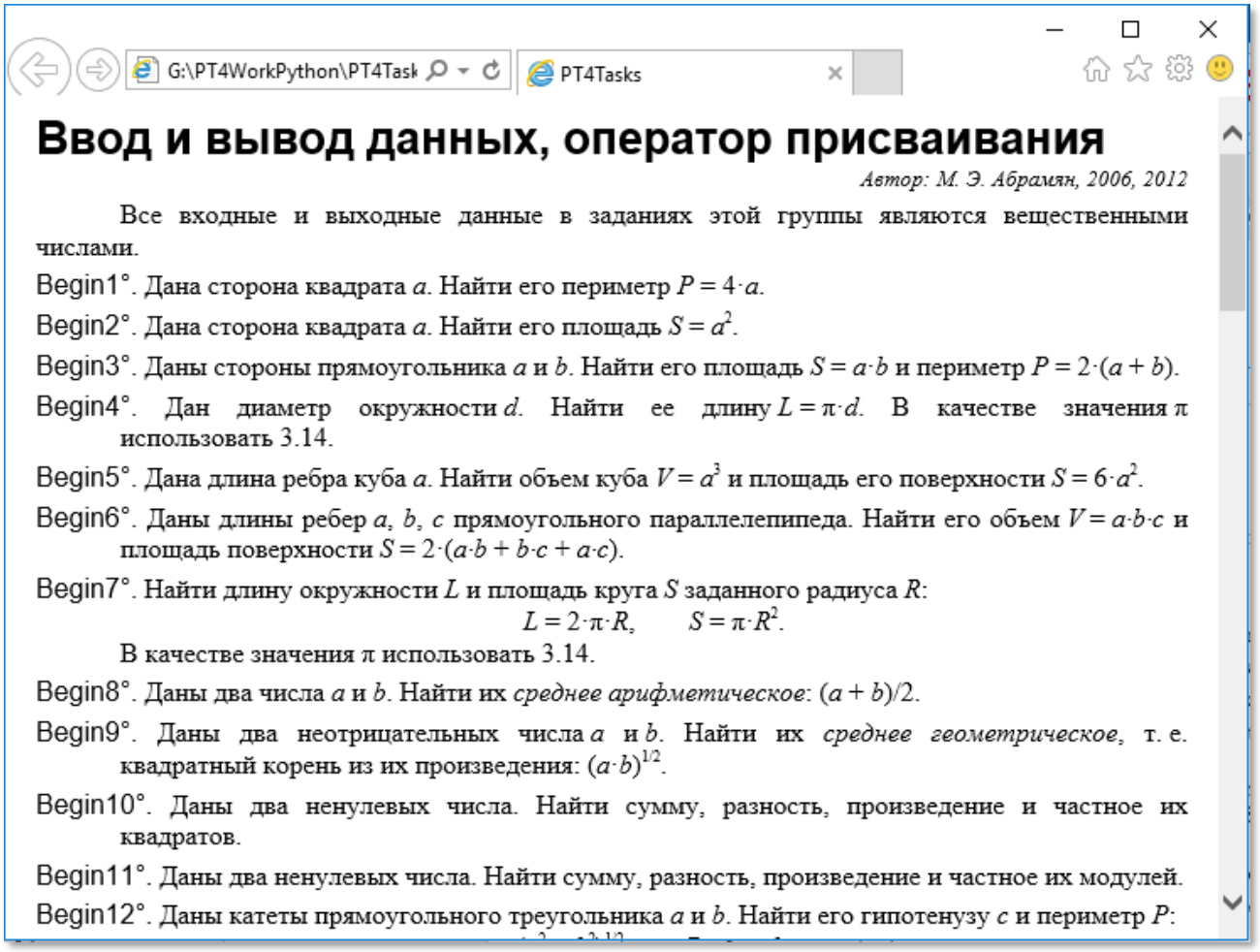


Рис. 5. Весь список

Решение второй задачи практически повторяет решение первой. Нужно только переименовать переменные и правильно записать формулу для вычисления площади квадрата.

В математической формуле используется операция возведения в квадрат. В *Питоне* для этого имеется операция возведения в степень **, но обычно **возведение в квадрат заменяют умножением**.

Код программы может быть таким:

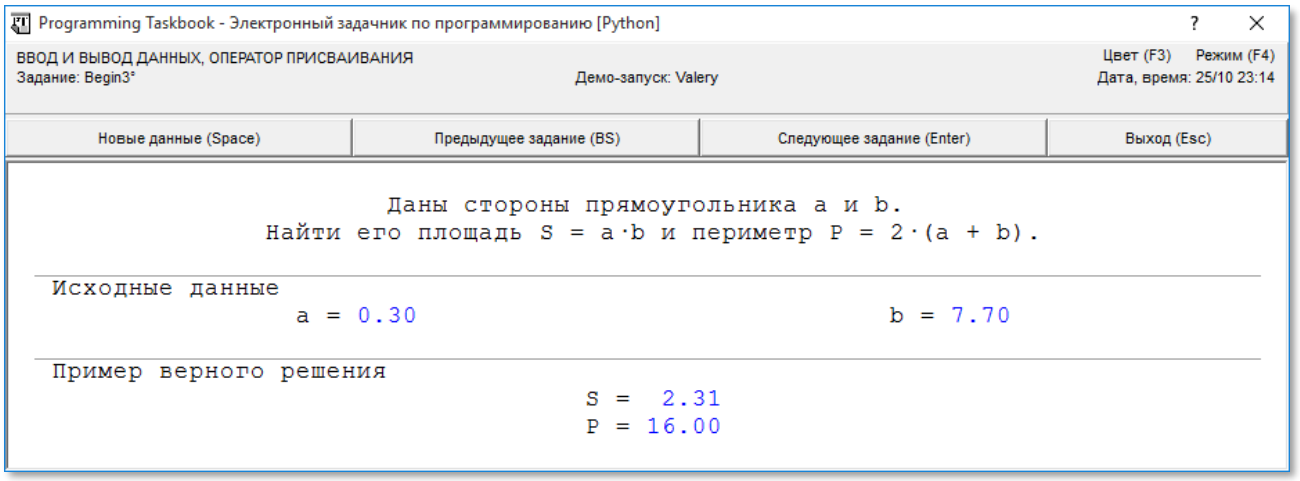
```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
```

```
    task("Begin2")

    #сторона квадрата:
    a = get_float()

    #площадь:
    s = a * a;

    #печатаем результат:
    put(s);

start(solve)
```

Кроме собственно вычисления площади квадрата, в нём нет ничего нового для вас. Лёгкая прогулка и приятная тренировка!

Задание Begin3

Получите заготовку для **третьего задания** и ознакомьтесь с его условием:



Задача посложнее прежних! Теперь нам нужно получить **2** значения – длину сторон прямоугольника **a** и **b**. Вычислить и напечатать также нужно **2** значения – площадь прямоугольника **s** и его периметр **p**.

В этом задании нам необходимы 2 значения:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
```

```
    task("Begin3")
    #стороны прямоугольника:
    a = get_float()
    b = get_float()
```

Вычисления и печать результатов выполняются точно так же, как и раньше:

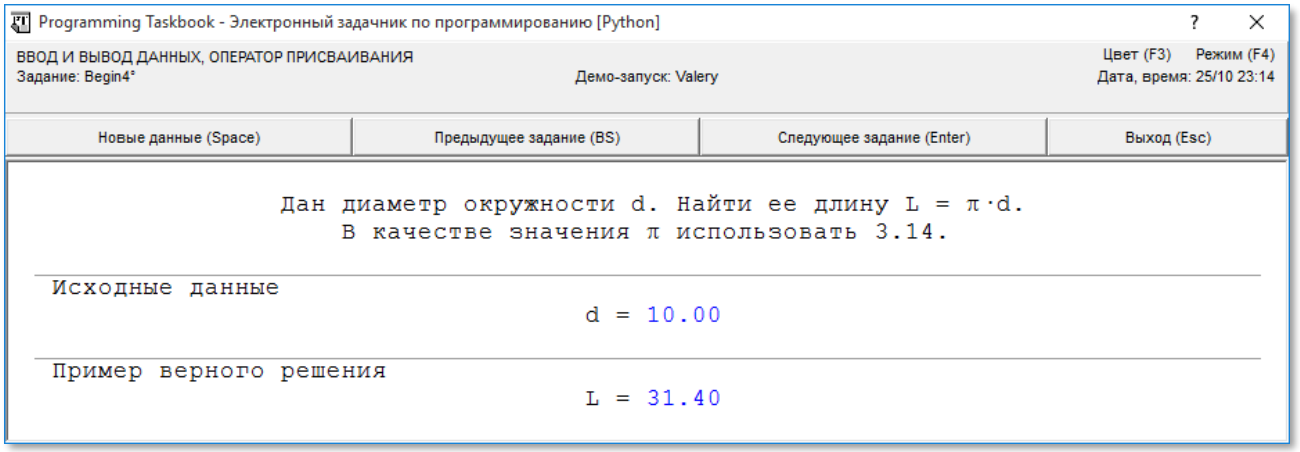
```
    #вычисляем площадь:
    s = a * b
    #печатаем значение:
    put(s)

    #вычисляем периметр:
    p = 2*(a + b)
    #печатаем значение:
    put(p)


start(solve)
```

Вычислять площадь и периметр можно в любом порядке, но напечатать их значения нужно именно так, как указано в условии задачи, – сначала площадь, а затем периметр, но не наоборот!

Замечание. Для проверки программы запустите её **несколько** раз. Ведь вполне вероятно, что при первом запуске результаты **случайно** совпадут с правильными, хотя само решения будет ошибочным.

Задание Begin4

Четвёртое задание практически повторяет первое:

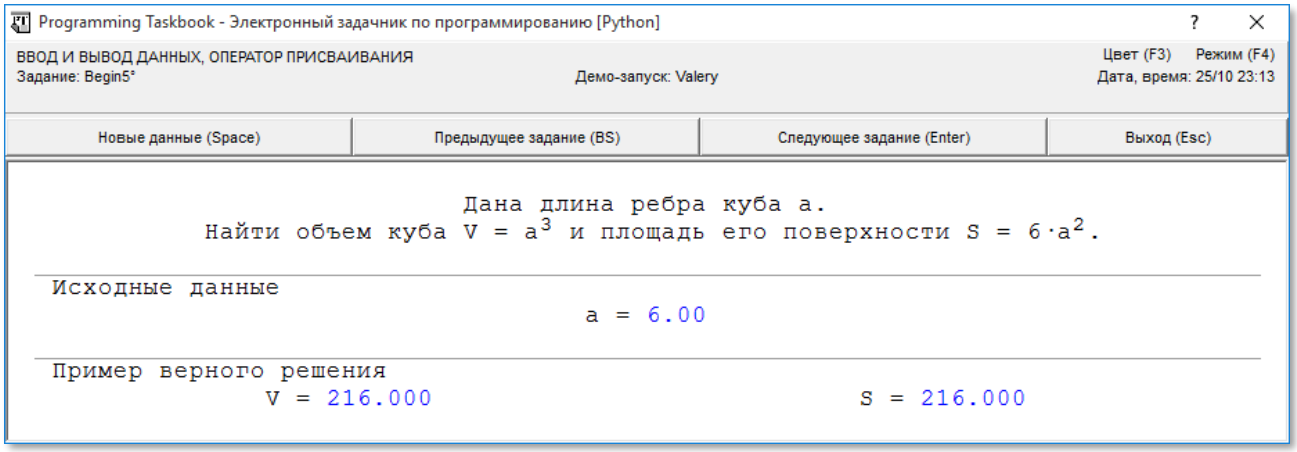


Ни обсуждать, ни комментировать здесь нечего:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin4")

    #диаметр:
    d = get_float()

    #длина окруужности:
    l = 3.14 * d
    put(l)


start(solve)
```

Задание Begin5

Переходим в пространство и решаем стереометрическую задачу:

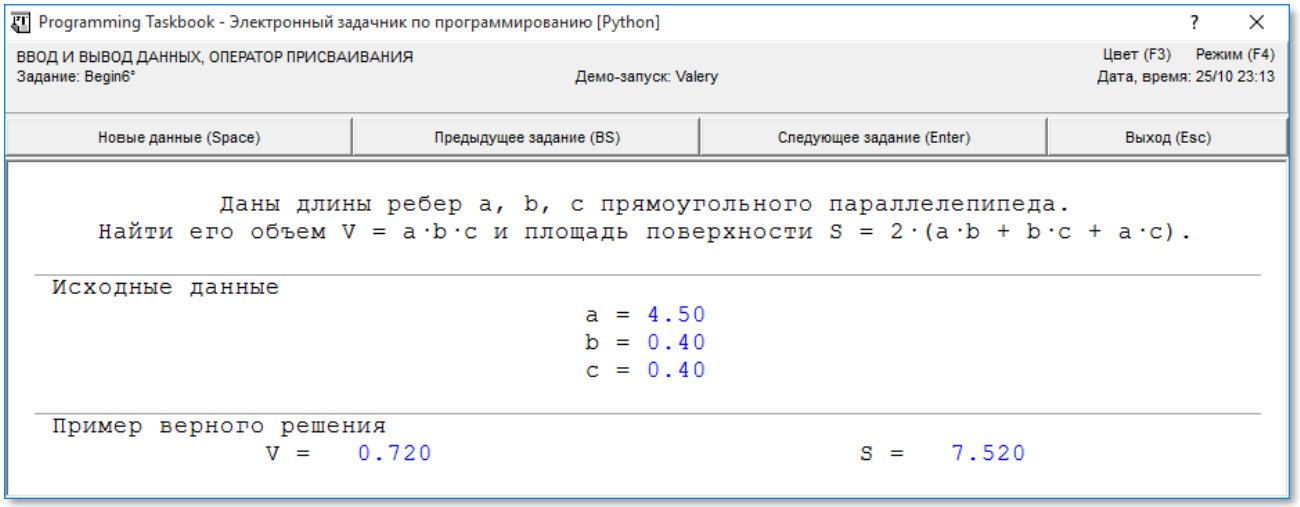


При вычислении объёма и площади поверхности куба заменяйте возведение в степень умножением. Других «хитростей» в этой задаче нет:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin5")

    #длина ребра куба:
    a = get_float()

    #объём:
    v = a * a * a # a**3
    put(v)

    #площадь поверхности:
    s = 6 * a * a
    put(s)


start(solve)
```

Задание Begin6

Переходим к следующему заданию:



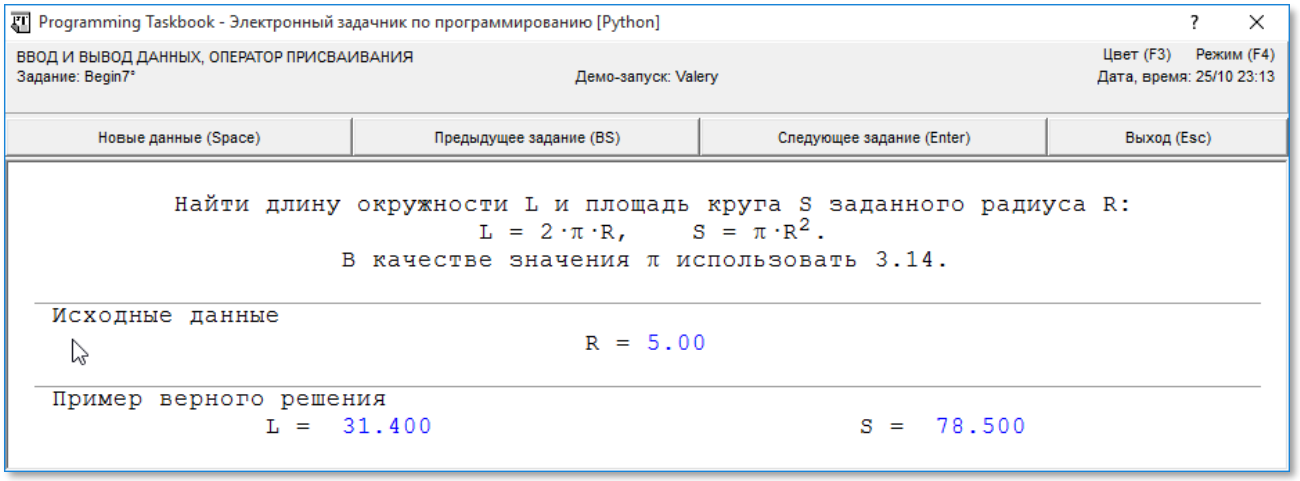
Самое сложное в этом задании – суметь выговорить *прямоугольный параллелепипед*. Смогли? – Тогда вычисления не доставят вам много хлопот!

Правда, нужно получить значения для **трёх** переменных, а формула для вычисления площади поверхности довольно замысловатая:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin6")

    #длины рёбер ПП:
    a = get_float()
    b = get_float()
    c = get_float()

    #объём:
    v = a*b*c
    put(v)

    #площадь поверхности:
    s = 2*(a*b + b*c + a*c)
    put(s)


start(solve)
```

Задание Begin7

Опять простенькое задание на вычисления по готовым формулам:

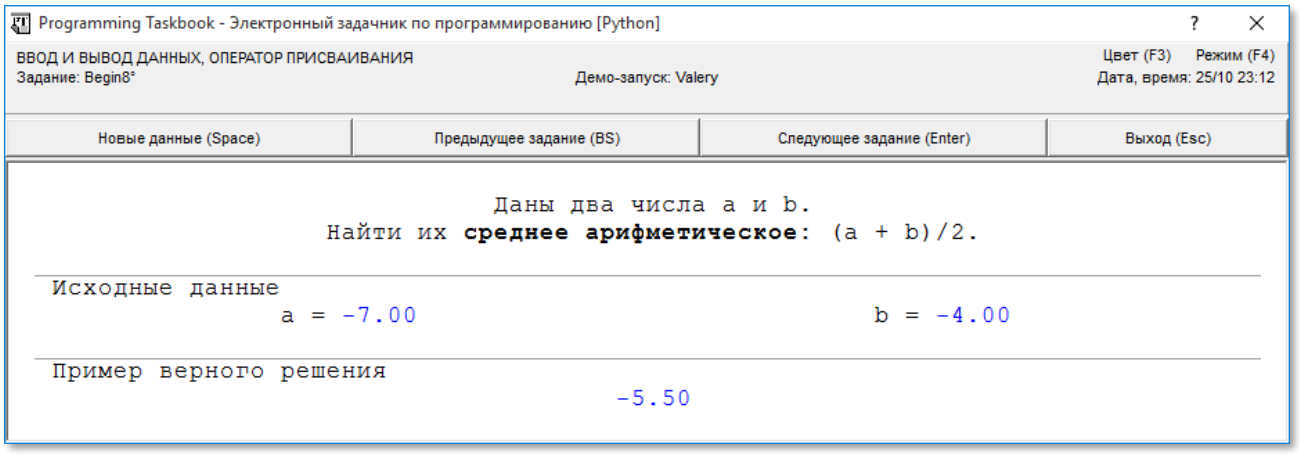


Все комментарии – в исходном тексте программы:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin7")

    #радиус:
    r = get_float()

    #длина окружности:
    l = 2*3.14*r
    put(l)

    #площадь круга:
    s = 3.14*r*r
    put(s)


start(solve)
```

Задание Begin8

Здесь нам впервые встречается операция **вещественного деления**, которая обозначается в исходном коде косой дробной чертой:



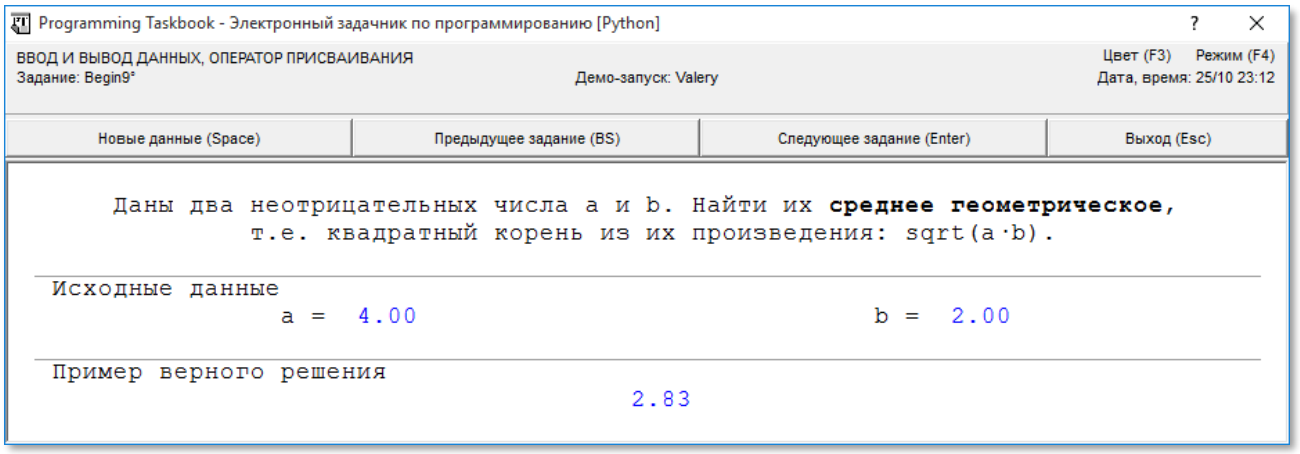
> Не путайте эту операцию с операцией целочисленного деления **//**!

Сам код программы не должен вызвать у вас никаких вопросов:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin8")

    #числа:
    a = get_float()
    b = get_float()

    #среднее арифметическое:
    sa = (a + b) / 2
    put(sa)


start(solve)
```

Задание *Begin9*

И опять новинка! Для вычисления среднего геометрического двух чисел необходимо извлечь **квадратный корень** из их произведения:

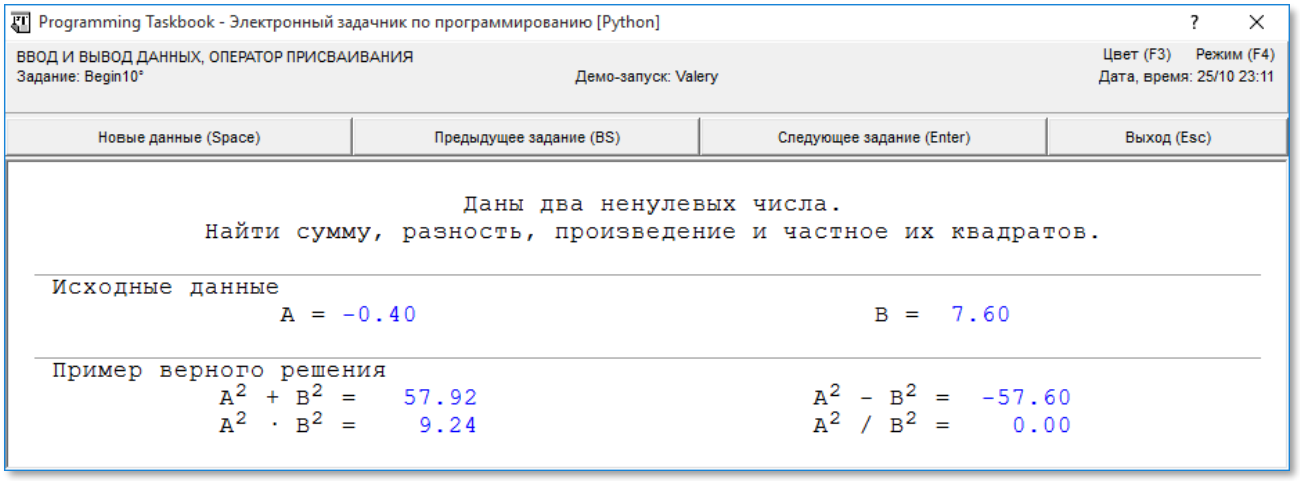


Для этого в *Питоне* имеется функция **sqrt**, которая возвращает корень квадратный из значения выражения в скобках. Так как она находится в модуле *math*, его следует импортировать:

```
# -*- coding: cp1251 -*-
from pt4 import *
import math

def solve():
    task("Begin9")

    #числа:
    a = get_float()
    b = get_float()

    #среднее геометрическое:
    sg = math.sqrt(a * b)
    put(sg)


start(solve)
```

Задание Begin10

В этом задании всего 2 числа, но они входят в четыре выражения:



Так как квадраты чисел **a** и **b** используются в вычислениях неоднократно, то их необходимо вычислить один раз, а затем вместо умножения чисел подставлять переменные **a2** и **b2**:

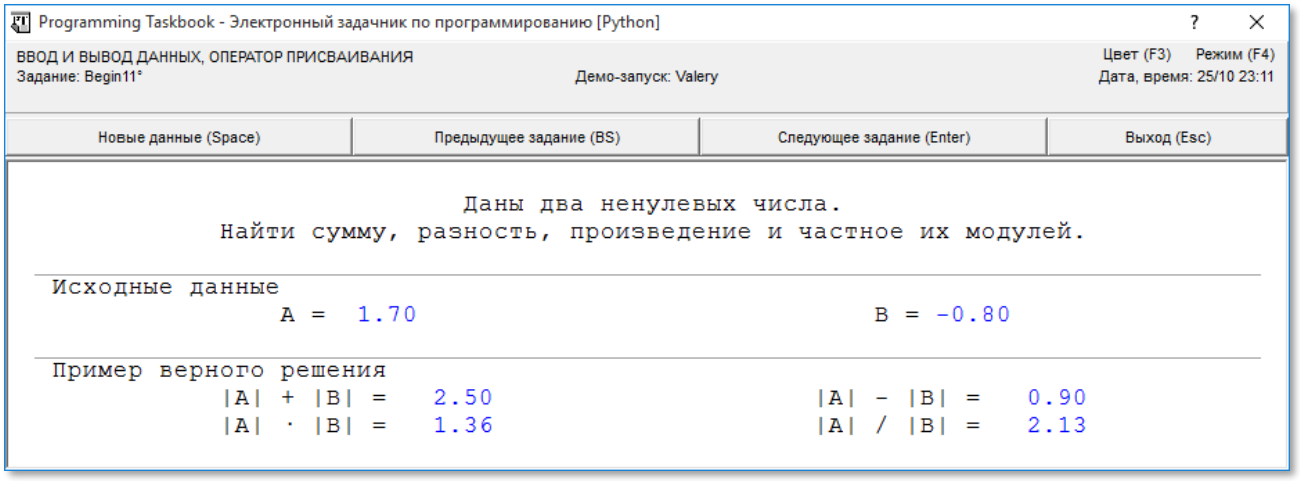
```
# -*- coding: cp1251 -*-

from pt4 import *
def solve():
    task("Begin10")

    #числа:
    a = get_float()
    b = get_float()

    #квадраты чисел:
    a2 = a*a
    b2 = b*b

    #сумма квадратов:
    sum = a2 + b2
    put(sum)

    #разность квадратов:
    razn = a2 - b2
    put(razn)

    #произведение квадратов:
    proiz = a2 * b2
    put(proiz)
```

```
    #частное от деления квадратов:
    chastn = a2 / b2
    put(chastn)


start(solve)
```

С этими заменами код становится более простым, понятным и надёжным. Всегда заменяйте повторяющиеся вычисления переменными, хранящими их значения!

Задание Begin11

Задача, аналогичная предыдущей, только вместо квадратов чисел нужно брать их **абсолютные значения**:



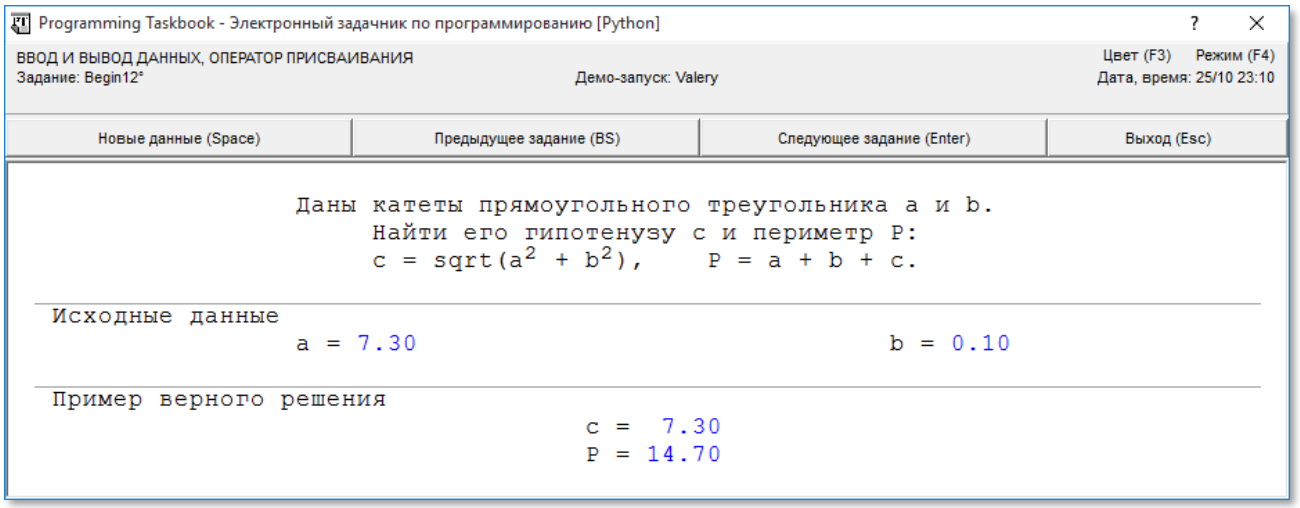
В *Питоне* имеется функция **abs**, которая возвращает абсолютное значение выражения в скобках. Для хранения абсолютных значений чисел **a** и **b** следует завести переменные **aa** и **ba**, которые затем использовать в вычислениях:

```
# -*- coding: cp1251 -*-
from pt4 import *
import math

def solve():
    task("Begin11")

    #числа:
    a = get_float()
```

```
    b = get_float()

    #модули чисел:
    aa = abs(a)
    ba = abs(b)

    #сумма модулей:
    sum = aa + ba
    put(sum)

    #разность модулей:
    razn = aa - ba
    put(razn)

    #произведение модулей:
    proiz = aa * ba
    put(proiz)

    #частное от деления модулей:
    chastn = aa / ba
    put(chastn)


start(solve)
```

Задание Begin12

Типичная задача из школьной геометрии:

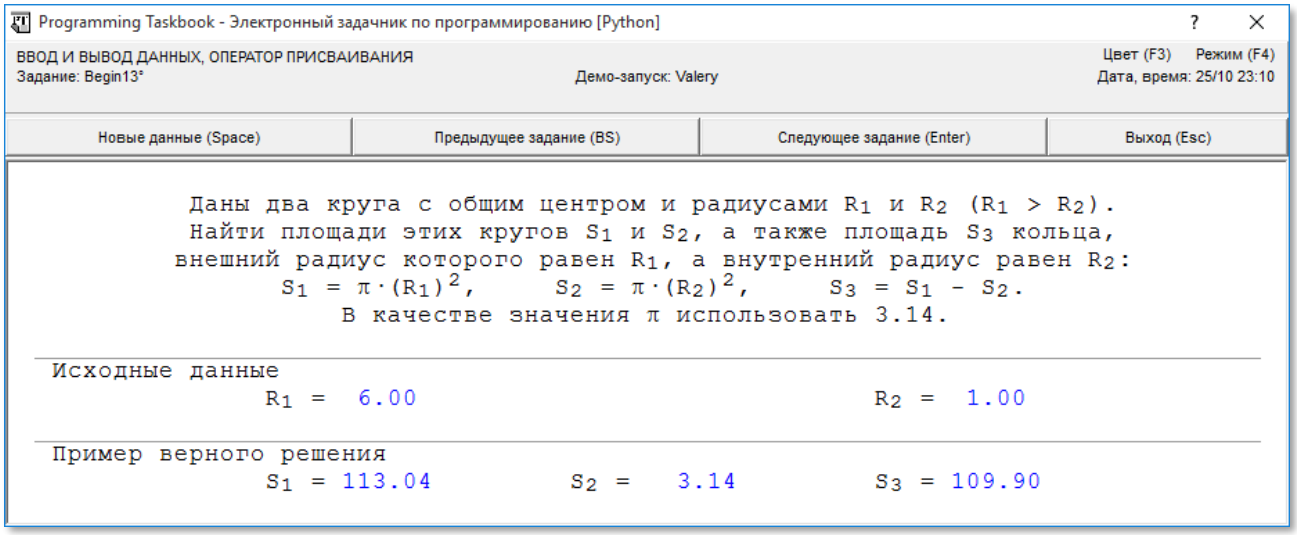


Так как в условии задачи уже даны готовые формулы, то нам не нужно даже вспоминать, чему равны пифагоровы штаны:

```python
# -*- coding: cp1251 -*-
from pt4 import *
import math

def solve():
    task("Begin12")

    #катеты:
    a = get_float()
    b = get_float()

    #длина гипотенузы:
    c = math.sqrt(a*a + b*b)
    put(c)

    #периметр треугольника:
    p = a + b + c
    put(p)


start(solve)
```

Задание Begin13

Условие задачи длинное, но легкоусвояемое:

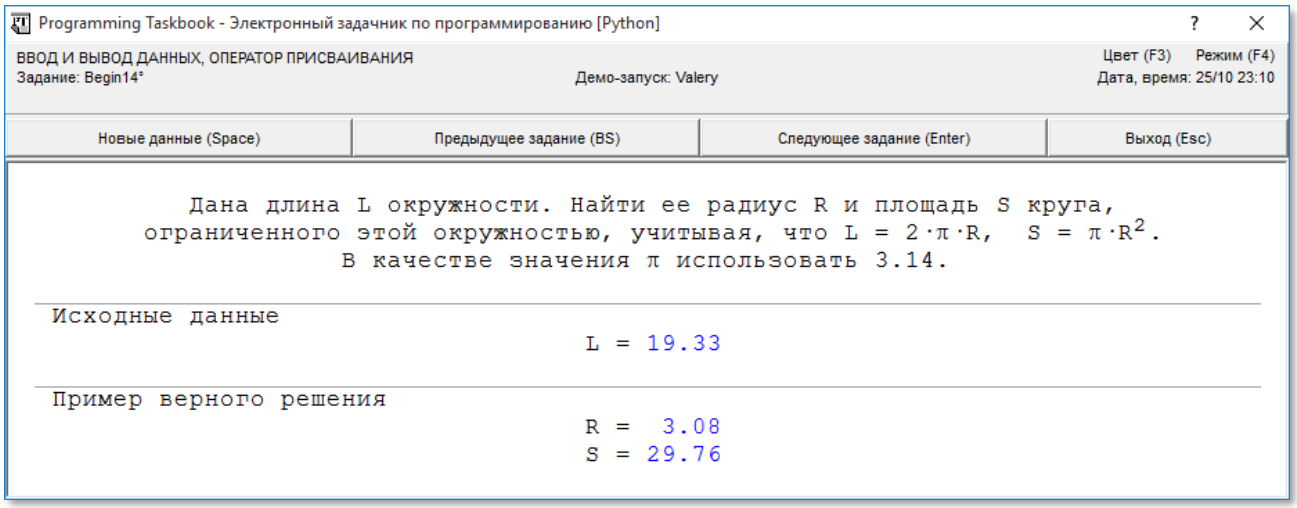


А вся программа – это простой перевод условия задачи на язык программирования:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin13")

    #радиусы кругов:
    r1 = get_float()
    r2 = get_float()

    #площадь первого круга:
    s1 = 3.14 * r1 * r1
    put(s1)

    #площадь второго круга:
    s2 = 3.14 * r2 * r2
    put(s2)

    #площадь кольца:
    s3 = s1 - s2
    put(s3)


start(solve)
```

Задание Begin14

А эта задача с хитрецой:



Радиус окружности не даётся в готовом виде – его нужно найти из формулы для длины окружности. Ну, это легко:

$L = 2 \pi R \rightarrow$

$R = L / 2 / \pi$

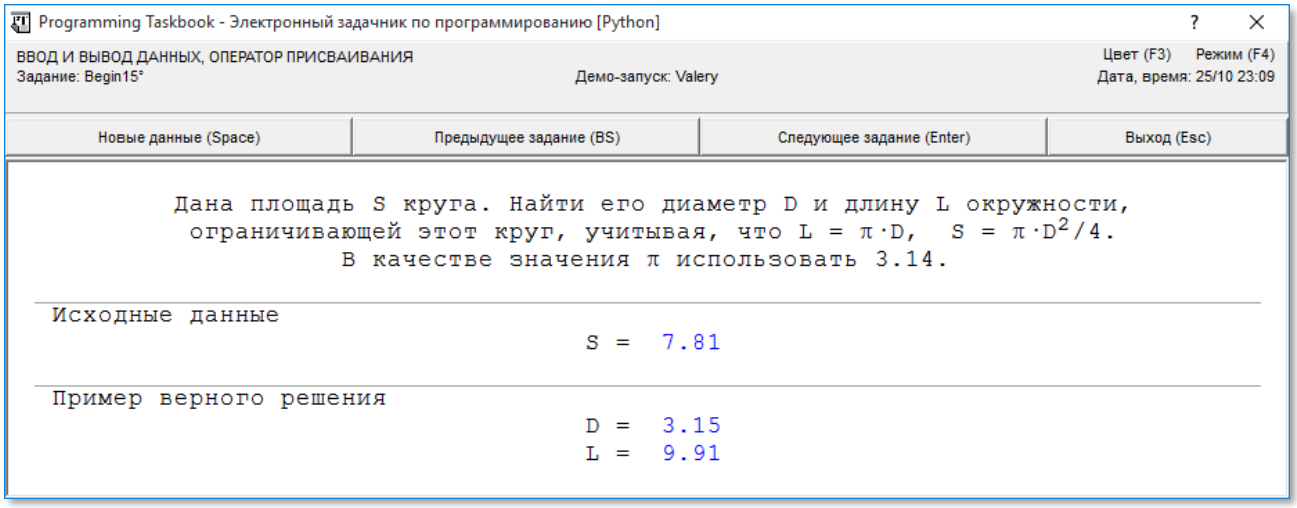
В знаменатель можно поставить и произведение *2π*, но лучше использовать двукратное деление.

Весь прочий код элементарен:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin14")

    #длина окружности:
    l = get_float()

    #радиус:
    r = l / 2 / 3.14
    put(r)

    #площадь круга:
    s = 3.14 * r * r
    put(s)


start(solve)
```

Задание Begin15

В этом задании нужно вычислить длину окружности, предварительно найдя из предложенной формулы её диаметр:



Если $S = \pi D^2/4$, то **диаметр равен**:

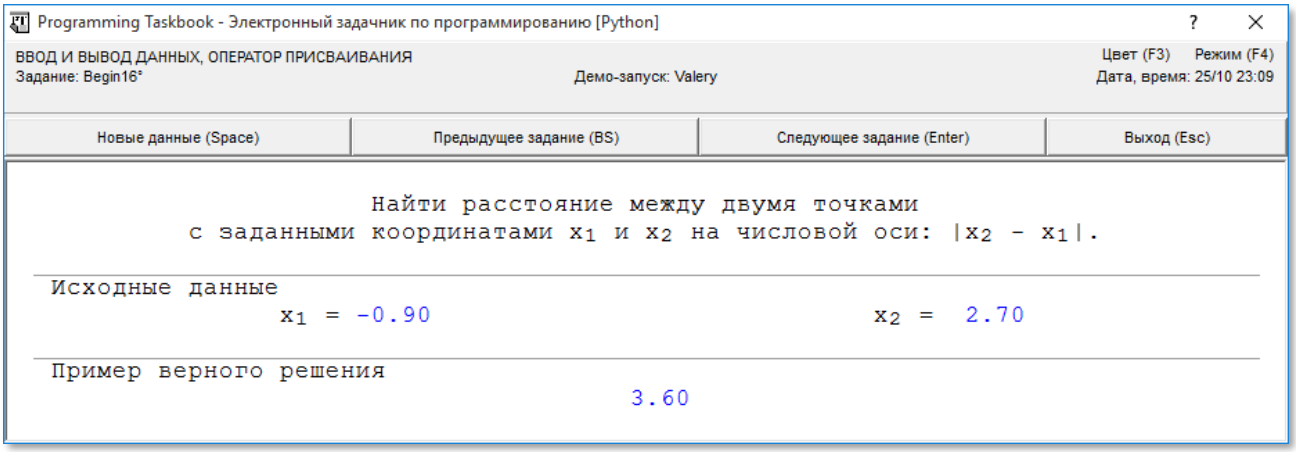
$$D = \sqrt{S * 4 / 3.14}$$

Остальное, как говорится, дело компьютерной техники:

```
# -*- coding: cp1251 -*-
from pt4 import *
import math

def solve():
    task("Begin15")

    #площадь круга:
    s = get_float()

    #диаметр:
    d = math.sqrt(s * 4 / 3.14)
    put(d)

    #длина окружности:
    l = 3.14 * d
    put(l)


start(solve)
```

Задание Begin16

Очень простое задание:

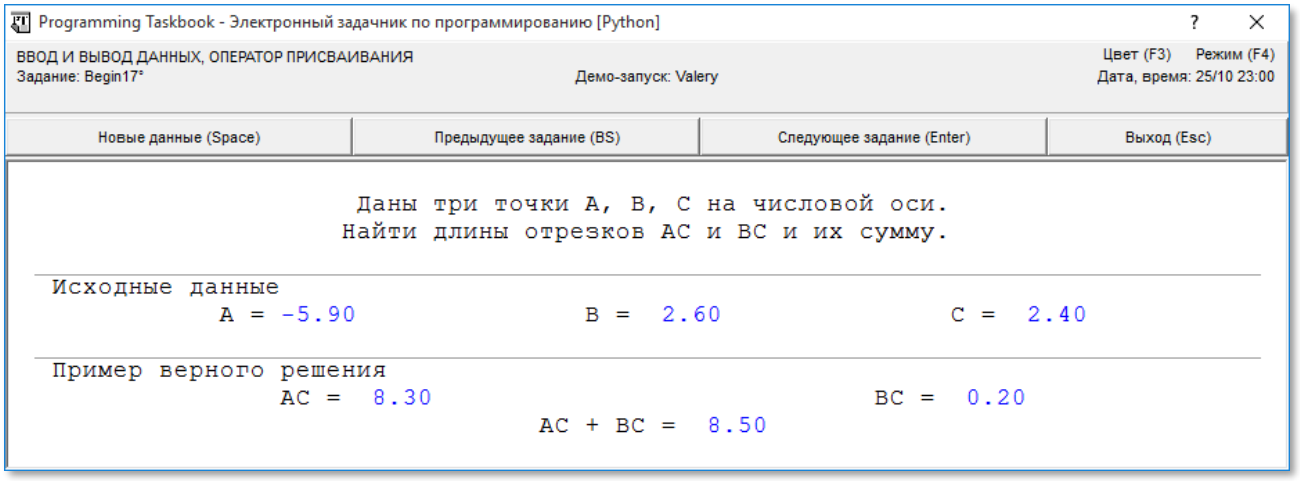


Расстояние между двумя точками на числовой оси равно абсолютной величине разности их координат:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin16")

    #координаты точек:
    x1 = get_float()
    x2 = get_float()

    #расстояние:
    l = abs(x1 - x2)
    put(l)


start(solve)
```

Задание Begin17

Развиваем плодотворную идею предыдущего задания:

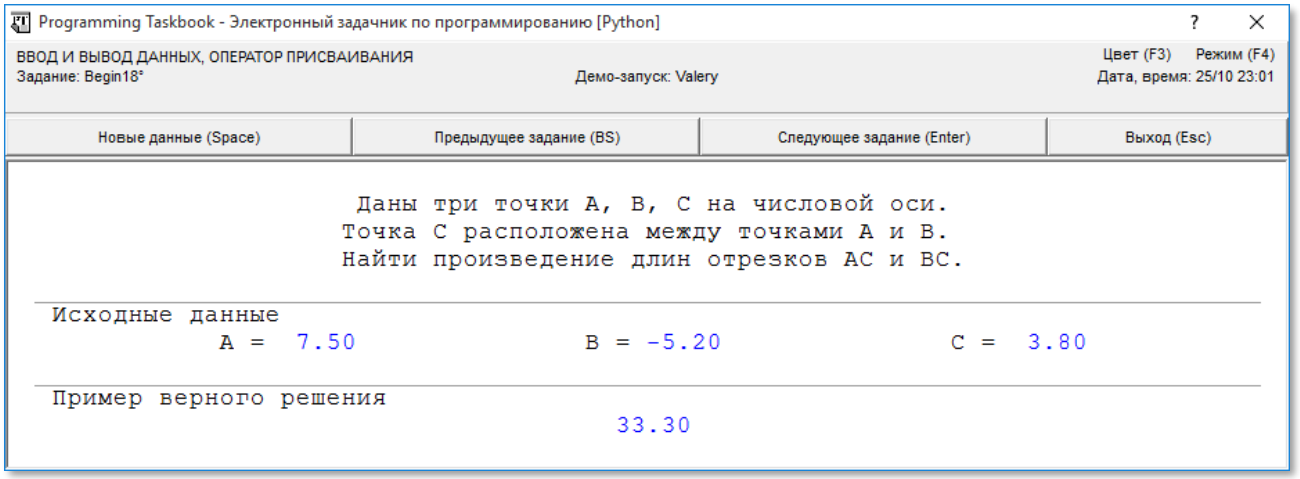


Длину отрезка на числовой оси вы уже научились вычислять. Поэтому найти длины двух отрезков, а затем их сумму проще простого:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin17")

    #координаты точек:
    a = get_float()
    b = get_float()
    c = get_float()

    #длина отрезка AC:
    ac = abs(a - c)
    put(ac)

    #длина отрезка BC:
    bc = abs(b - c)
    put(bc)

    #сумма длин отрезков:
    sum = ac + bc
    put(sum)


start(solve)
```

Задание Begin18

Опять нам нужно найти длины двух отрезков:



Чтобы не ошибиться в расположении точек на оси, «нарисуем» их:

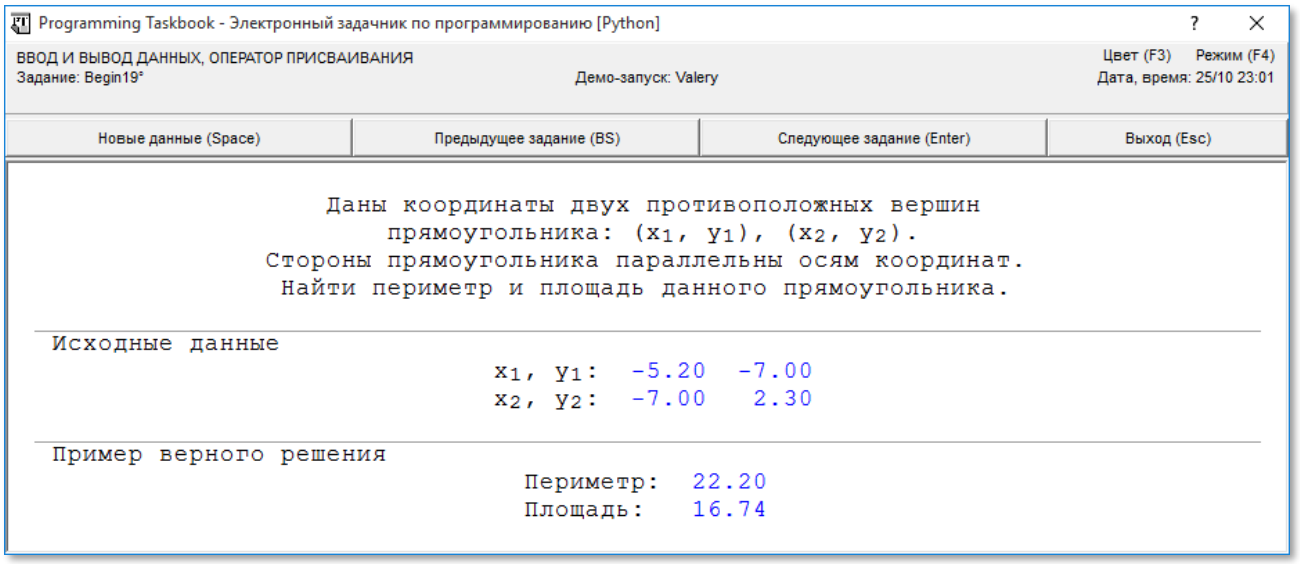
A----C----B
B----C----A

Таким образом, мы должны вычислить длину отрезков **AC** и **CB**, то есть абсолютную величину разности их координат:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin18")

    #координаты точек:
    a = get_float()
    b = get_float()
    c = get_float()

    #длина отрезка AC:
    ac = abs(a - c)

    #длина отрезка BC:
    bc = abs(b - c)

    #произведение длин отрезков:
    proizv = ac * bc
    put(proizv)


start(solve)
```

Задание Begin19

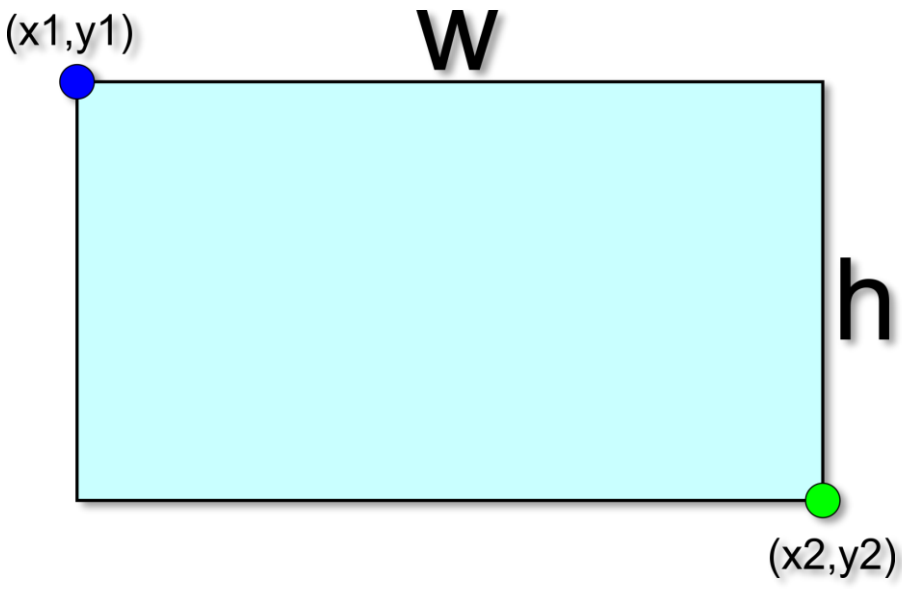
Площадь и периметр прямоугольника вы умеете вычислять, но для этого нужно знать **длину** его сторон:



В данном случае мы можем считать длину одной стороны прямоугольника его шириной **w**, а длину второй стороны – его высотой **h**.

Ширина прямоугольника равна абсолютной величине разности горизонтальных координат его противоположных вершин, а высота – абсолютной величине разности вертикальных координат.

Начертите прямоугольник и подпишите координаты его противоположных вершин, и тогда эти утверждения станут очевидными:

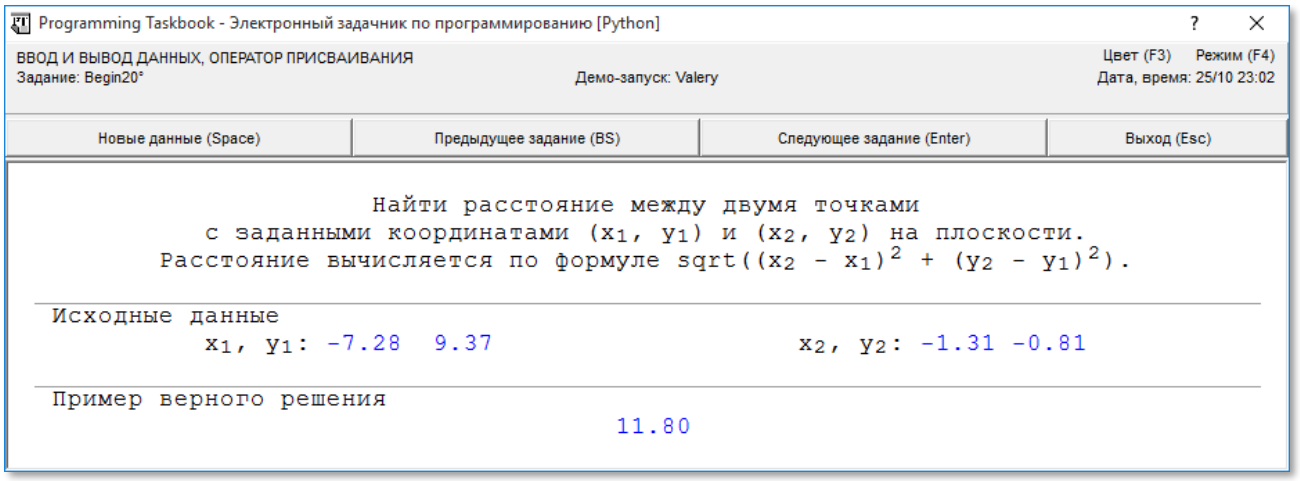


После этих предварительных рассуждений задача практически решена:

```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin19")

    # координаты вершин:
    x1 = get_float()
    y1 = get_float()
    x2 = get_float()
    y2 = get_float()

    #ширина прямоугольника:
    w = abs(x1 - x2)

    #высота прямоугольника:
    h = abs(y1 - y2)

    #периметр прямоугольника:
    p = 2 * (w + h)
    put(p)

    #площадь прямоугольника:
    s = w * h
    put(s)


start(solve)
```

Задание Begin20

Решение этой задачи сводится к переводу её условия на язык программирования *Питон:*

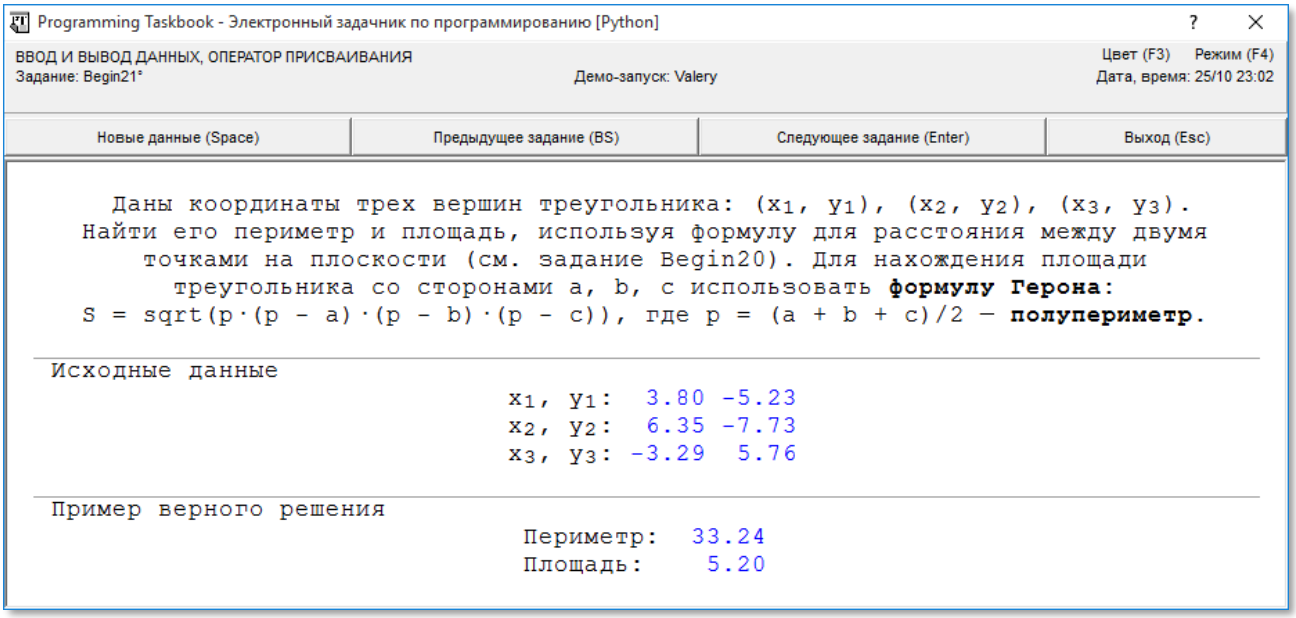


Действительно, координаты точек вы получаете от программы, а формула для вычисления расстояния между ними дана в готовом виде:

```
# -*- coding: cp1251 -*-
from pt4 import *
import  math

def solve():
    task("Begin20")

    # координаты вершин:
    x1 = get_float()
    y1 = get_float()
    x2 = get_float()
    y2 = get_float()

    #разность координат по осям X и Y:
    dx = x1 - x2
    dy = y1 - y2

    #расстояние между двумя точками:
    d = math.sqrt(dx * dx + dy * dy)
    put(d)


start(solve)
```

Задание Begin21

Число точек на плоскости увеличивается:



Усложняет ли это нашу задачу? – Ничуть!

Чтобы сделать код более наглядным, мы предварительно находим разность координат заданных точек по обеим осям и сохраняем значения в переменных *dx1, dy1, dx2, dy2, dx3, dy3*:

```
# -*- coding: cp1251 -*-
from pt4 import *
import math

def solve():
    task("Begin21")

    # координаты вершин:
    x1 = get_float()
    y1 = get_float()
    x2 = get_float()
    y2 = get_float()
    x3 = get_float()
    y3 = get_float()

    #разность координат по осям X и Y:
    dx1 = x1 - x2
    dy1 = y1 - y2
    dx2 = x1 - x3
    dy2 = y1 - y3
```

```
    dx3 = x3 - x2
    dy3 = y3 - y2
```

Расстояния между парами точек (это длины сторон треугольника) также следует вычислить отдельно и сохранить в переменных *a, b, c:*

```
#расстояния между парами точек:
a = math.sqrt(dx1 * dx1 + dy1 * dy1)
b = math.sqrt(dx2 * dx2 + dy2 * dy2)
c = math.sqrt(dx3 * dx3 + dy3 * dy3)
```

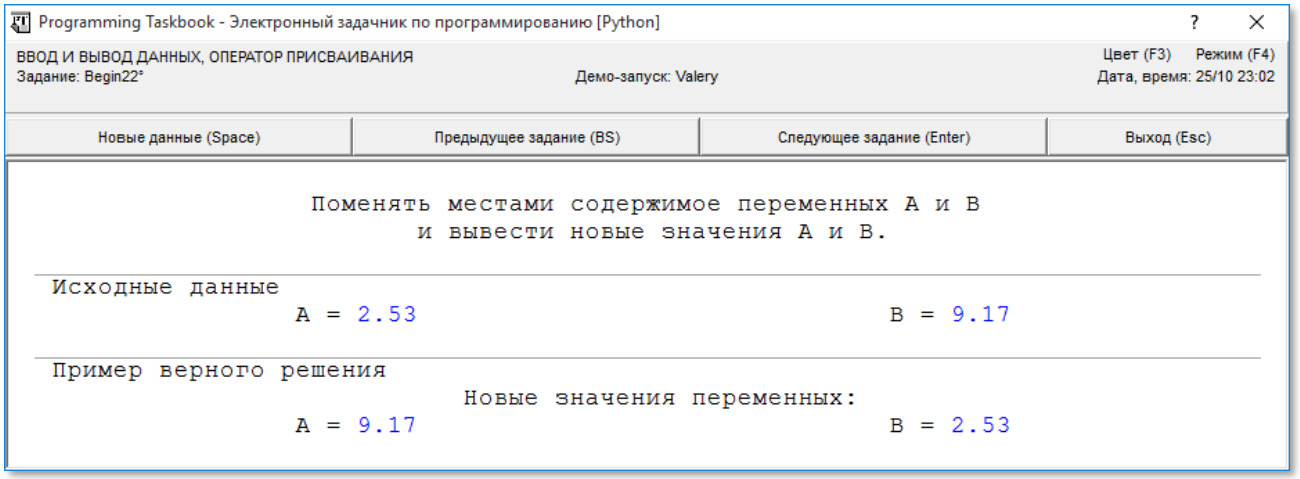
При вычислении расстояний мы **дважды** используем вычисленные ранее переменные. Согласитесь, такой код гораздо понятнее, чем если бы мы записывали вместо них разности координат в скобках.

Остальная часть кода просто повторяет условие задачи на языке *Питон*:

```
    #периметр треугольника:
    p = a + b + c
    put(p)

    #полупериметр треугольника:
    p2 = p / 2

    #площадь треугольника:
    s = math.sqrt(p2 * (p2 - a) * (p2 - b) * (p2 - c))
    put(s)


start(solve)
```

Задание Begin22

Обмен значениями – типовая задача в программировании:



Хитрость этой операции заключается в том, что в результате выполнения очевидного кода:

```
a = b
b = a
```

обмена значениями не произойдёт!

После выполнения первого оператора присваивания переменная **a** получит значение переменной **b**. Поэтому второй оператор не изменит значение переменной **b**, поскольку переменная **a** уже получила то же самое значение **b**.

Таким образом, вместо обмена значениями обе переменные будут иметь одинаковое значение *b*.

В таких случаях всегда используется **вспомогательная переменная tmp**, в которой необходимо сохранить значение одной из переменной. В нашем примере – переменной **a**. Тогда второй оператор присваивания должен быть таким:

```
b = tmp
```

К этому времени переменная **a** получила значение переменной **b**, но в переменной **tmp** сохранилось её значение, и переменная **b** получит исходное значение переменной **a**. Обмен состоялся!

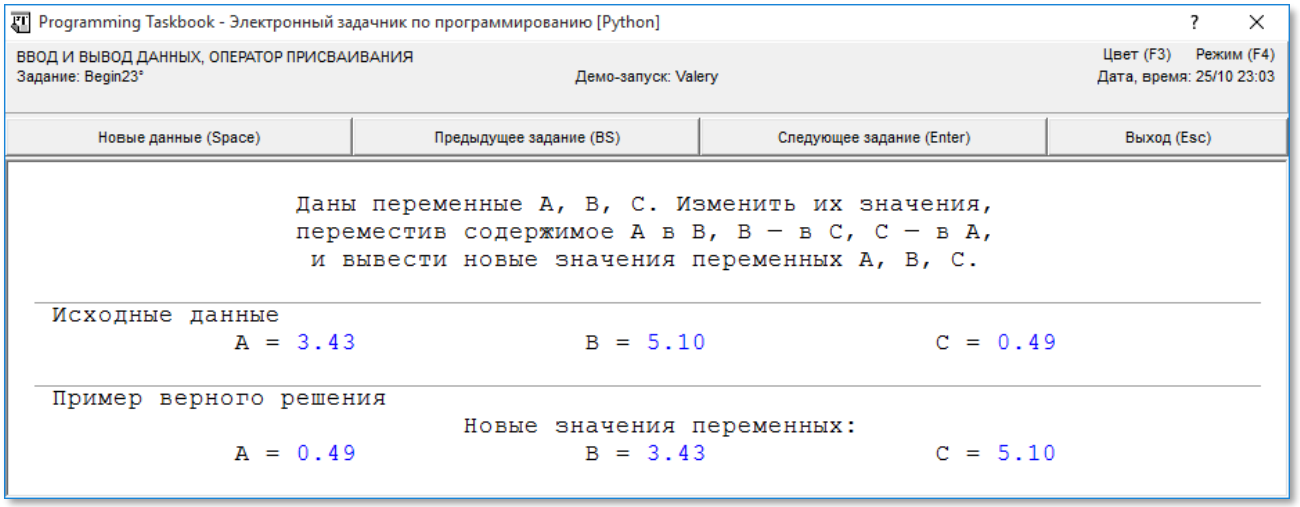
```
# -*- coding: cp1251 -*-
from pt4 import *

def solve():
    task("Begin22")

    #значения переменнных:
    a = get_float()
    b = get_float()

    #вспомогательная переменная:
    tmp = a
    #меняем значения переменных:
    a = b
    b = tmp

    #новые значения переменных:
    put(a, b)


start(solve)
```

Именно так обменивают значения переменных во многих языках программирования, но в *Питоне* можно обойтись без вспомогательной переменной! Тогда вся задача решается одной строкой:

```
a, b = b, a
```

Задание Begin23

Это задание – усложнённый вариант предыдущего:



Нам предстоит **циклически** изменить значения переменных:

A → B → C → A

Если мы будем действовать «очевидным» способом, то все переменные получат значение переменной **a**:

```
b = a
c = b
a = c
```

Как и при любом обмене, нам необходим посредник, то есть вспомогательная переменная **tmp**. В неё можно временно поместить значение какой-нибудь переменной, например, **a**.

Теперь мы можем безнаказанно изменить значение переменной **a**, то есть присвоить ей значение переменной **c**. Это легко понять, если посмотреть на цикл обмена значениями, который мы записали выше. Если мы запомнили значение **первой** переменной, то присваивать новые значения переменным нужно **от конца к началу** этого цикла:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin23")

    #значения переменных:
```

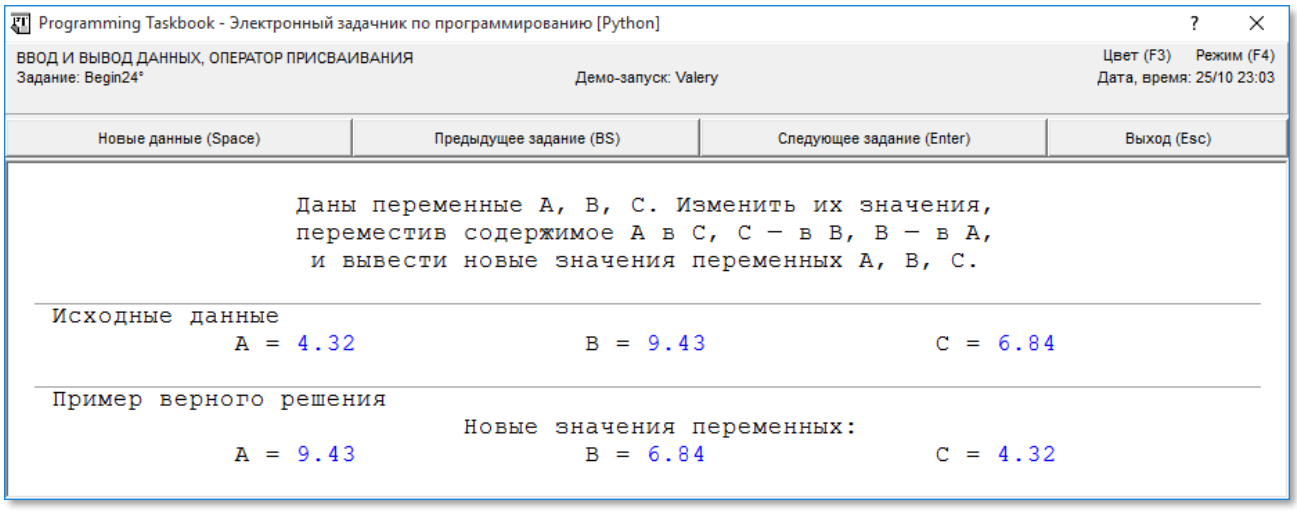
```
    a = get_float()
    b = get_float()
    c = get_float()

    #вспомогательная переменная:
    tmp = a
    #меняем значения переменных:
    a = c
    c = b
    b = tmp

    #новые значения переменных:
    put(a, b, c)


start(solve)
```

«Питоновский» способ избавляет нас от необходимости думать, но требует внимания при записи обменной операции:

```
a, b, c = c, a, b
```

Задание Begin24

Практически то же самое задание, что и предыдущее:



Чтобы нас запутать, порядок обмена значениями изменён:

A → C → B → A

Но, пользуясь этой записью цикла, мы легко решим задачу:

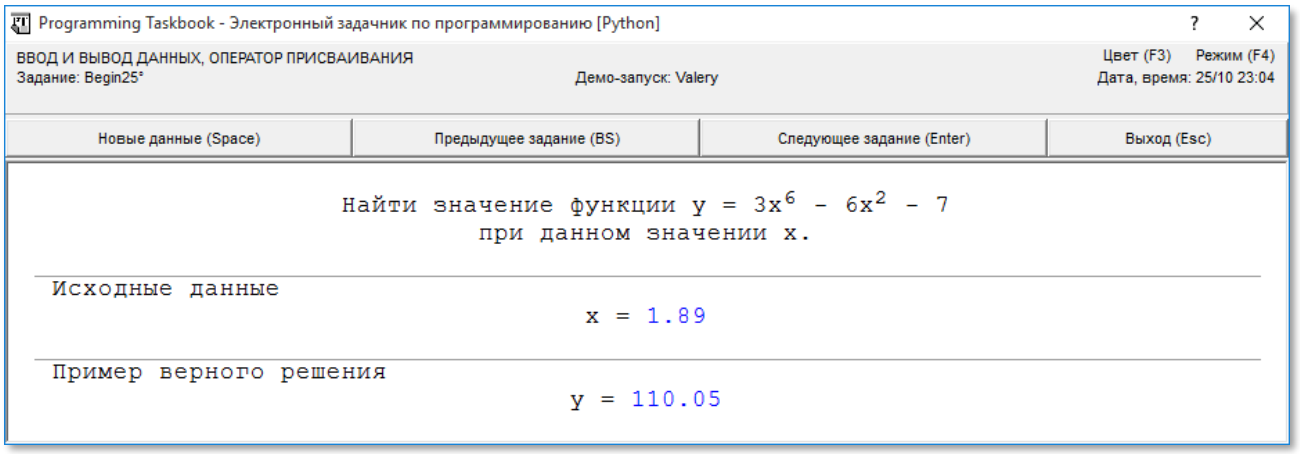
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin24")

    #значения переменнных:
    a = get_float()
    b = get_float()
    c = get_float()

    #вспомогательная переменная:
    tmp = a
    #меняем значения переменных:
    a = b
    b = c
    c = tmp

    #новые значения переменных:
    put(a, b, c)


start(solve)
```

А вот решение задачи на «чистом» *Питоне*:

```
a, b, c = b, c, a
```

Задание Begin25

Возвращаемся к вычислениям по формулам:



Понятно, что вычислить значение функции очень просто, поэтому в задаче предполагается, что это нужно сделать **рационально**, то есть с минимальным числом умножений.

Очевидный способ вычисления значения функции такой:

```
y = 3*x**6 - 6*x**2 - 7
```

Здесь мы воспользовались операцией возведения в степень **. Однако в программировании целые степени обычно вычисляются иначе – «самоумножением» числа. Понятно, что вторую степень можно получить только так:

```
x2 = x * x
```

А вот для вычисления **шестой** степени не нужно 6 раз умножать число *x* само на себя, ведь у нас уже имеется вторая степень переменной *x*. Это значит, что шестую степень можно получить, умножив число **x2** само на себя всего 2 раза.

Всё решение может быть таким:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin25")
```

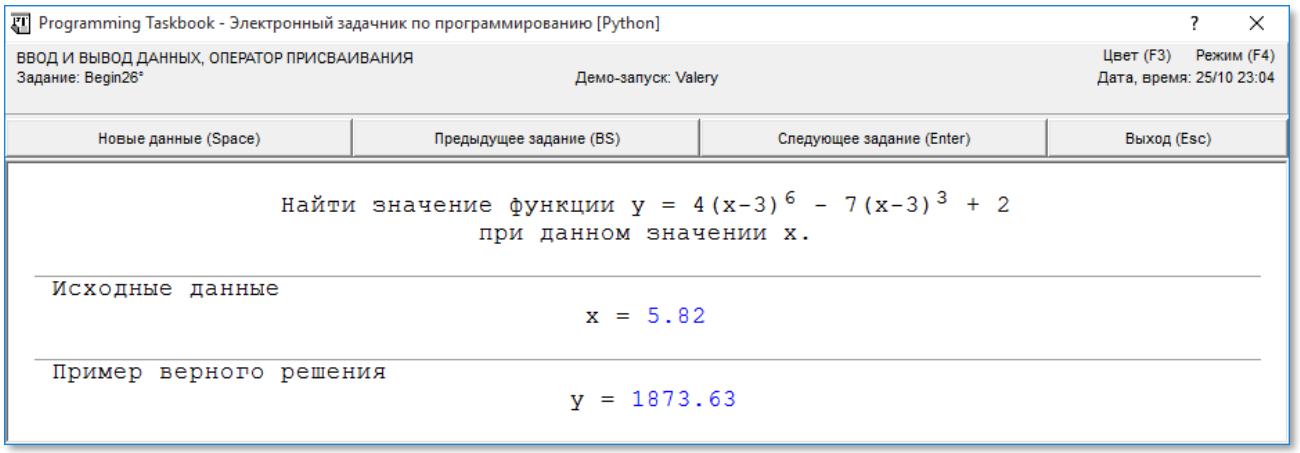
```
    #значение переменной:
    x = get_float()

    #x в квадрате:
    x2 = x * x #x**2

    #значение функции:
    y = 3 * x2 * x2 * x2 - 6 * x2 - 7
    #y = 3*x**6 - 6*x**2 - 7
    put(y)


start(solve)
```

Если немного подумать, то можно ещё более упростить вычисления:

```
y = x2 * (3 * x2 * x2 - 6) - 7
```

Задание Begin26

Усложнённый вариант предыдущего задания:



Так как в скобках стоит разность, то её лучше заменить новой переменной **x2**. Эта замена сократит код и сделает его более наглядным:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin26")
```

```
    #значение переменной:
    x = get_float()

    #x-3:
    x2 = x - 3
```

Куб разности проще всего найти двукратным умножением и сохранить в переменной **x3**:

```
#x-3 в кубе:
x3 = x2 * x2 * x2
```

Теперь вычисление значения функции стало простой задачей:

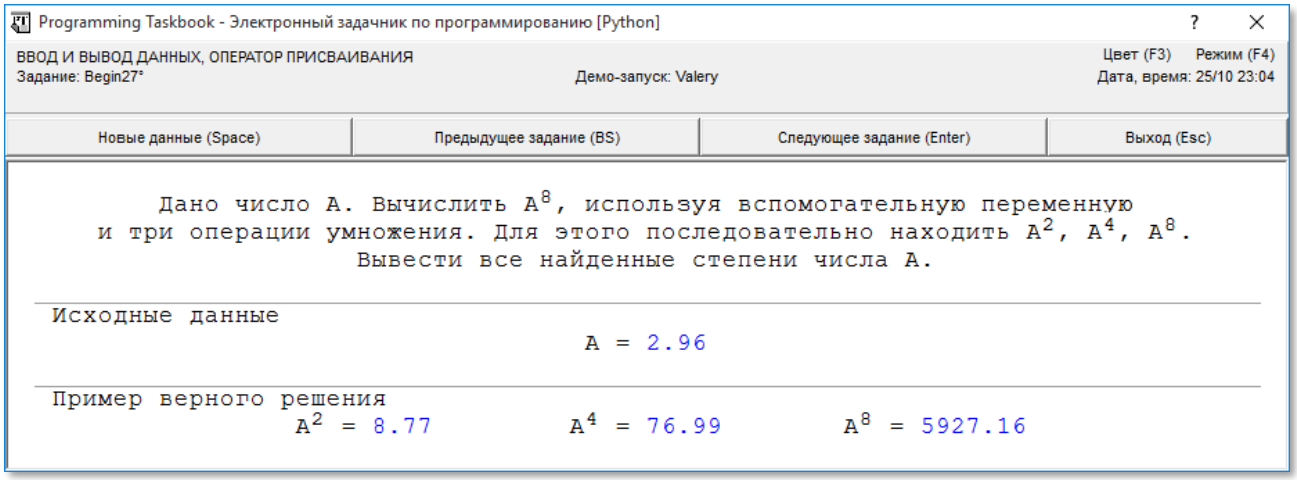
```
    #значение функции:
    y = 4 * x3 * x3 - 7 * x3 + 2
    put(y)


start(solve)
```

Можно преобразовать выражение для вычисления значения функции:

```
y = x3*(4 * x3 - 7) + 2
```

Задание Begin27

Это задание подтверждает правильность наших предшествующих решений:

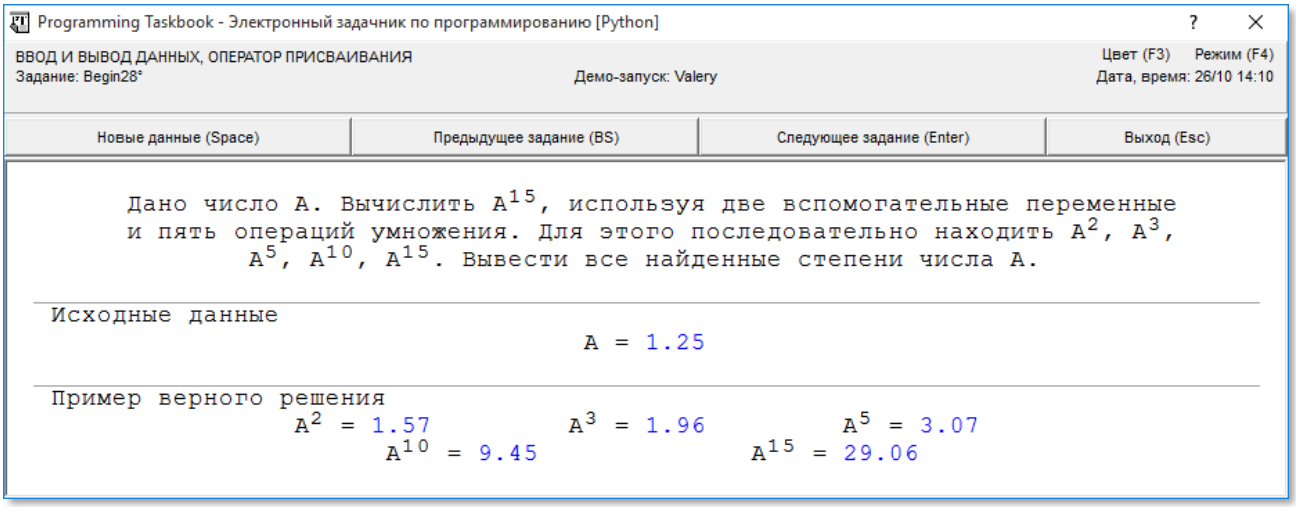


Если вы поняли, как вычисляются целые степени, то эта задача – лёгкая тренировка для вас:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin27")

    #число:
    a = get_float()

    #a в квадрате:
    a2 = a * a
    put(a2)

    #a в четвёртой степени:
    a4 = a2 * a2
    put(a4)

    #a в восьмой степени:
    a8 = a4 * a4
    put(a8)


start(solve)
```

Задание Begin28

Слегка усложнённый вариант предыдущего задания:



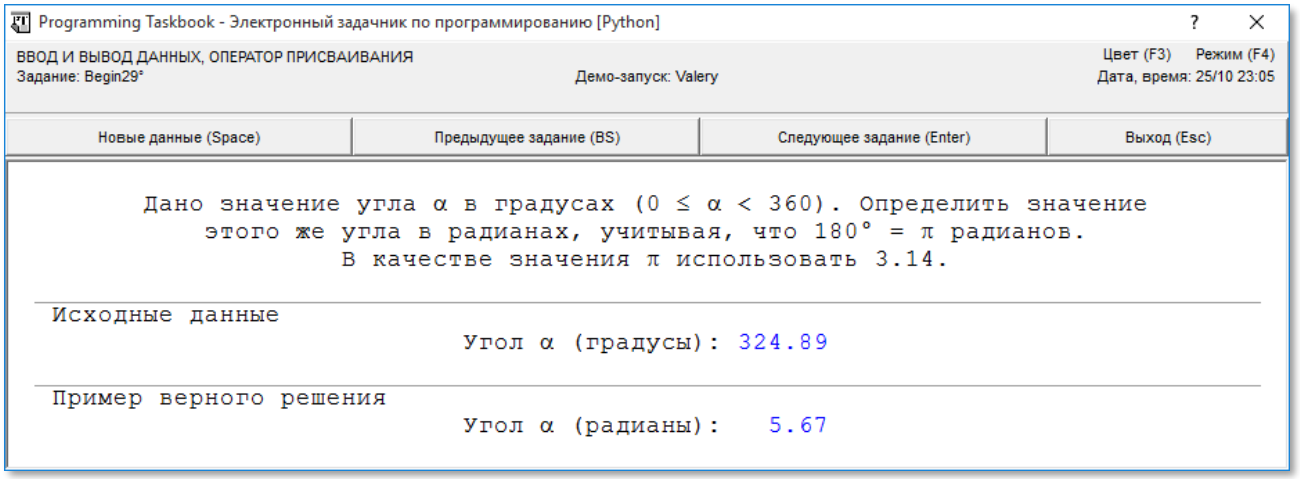
Его решение прямо указано в условии. Следуйте ему – и всё у вас получится!

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin28")

    #число:
    a = get_float()

    #a в квадрате:
    a2 = a * a
    put(a2)

    #a в кубе:
    a3 = a2 * a
    put(a3)

    #a в пятой степени:
    a5 = a2 * a3
    put(a5)

    #a в десятой степени:
    a10 = a5 * a5
    put(a10)

    #a в пятнадцатой степени:
    a15 = a10 * a5
    put(a15)
```

```
start(solve)
```

Задание Begin29

Простейшее задание на **пропорцию**:



Из условия нам известно, что 180° = π радианов. Тогда углу **alpha** в градусах соответствует угол **rad** в радианах:

180 – π
alpha – rad

Откуда:

rad = alpha / 180 * π

Записываем это уравнение на компьютерном языке:

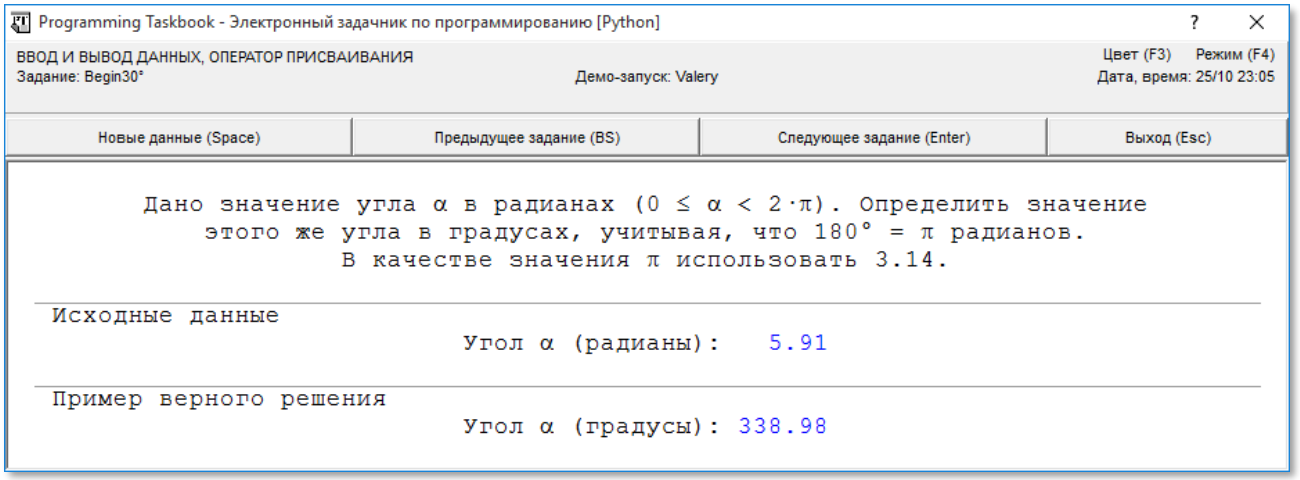
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin29")

    #угол в градусах:
    alpha = get_float()

    #угол в радианах:
    rad = alpha / 180.0 * 3.14
    put(rad)
```

```
start(solve)
```

Так как в правой части оператора присваивания находятся вещественные значения, то правильнее делить не на 180 – целочисленный литерал, а на 180.0 – вещественный литерал.

Задание Begin30

Та же самая задача, что и предыдущая, только наоборот:



Теперь мы должны найти значение угла в градусах, зная его значение в радианах.

Формула для вычисления угла в градусах легко выводится из той пропорции, которую мы написали в предыдущем задании:

alpha = rad * 180 / 3.14

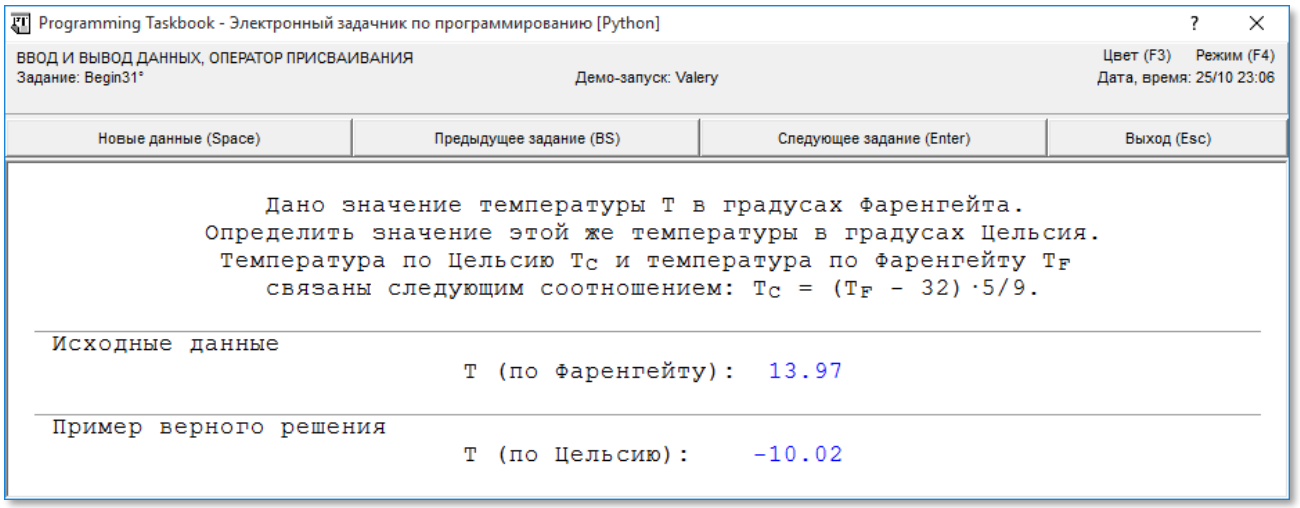
Про код и вовсе говорить нечего:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin30")

    #угол в радианах:
    rad = get_float()
```

```
    #угол в градусах:
    alpha = rad * 180.0 / 3.14
    put(alpha)


start(solve)
```

Задание Begin31

Наипростейшее задание на вычисления по готовой формуле:

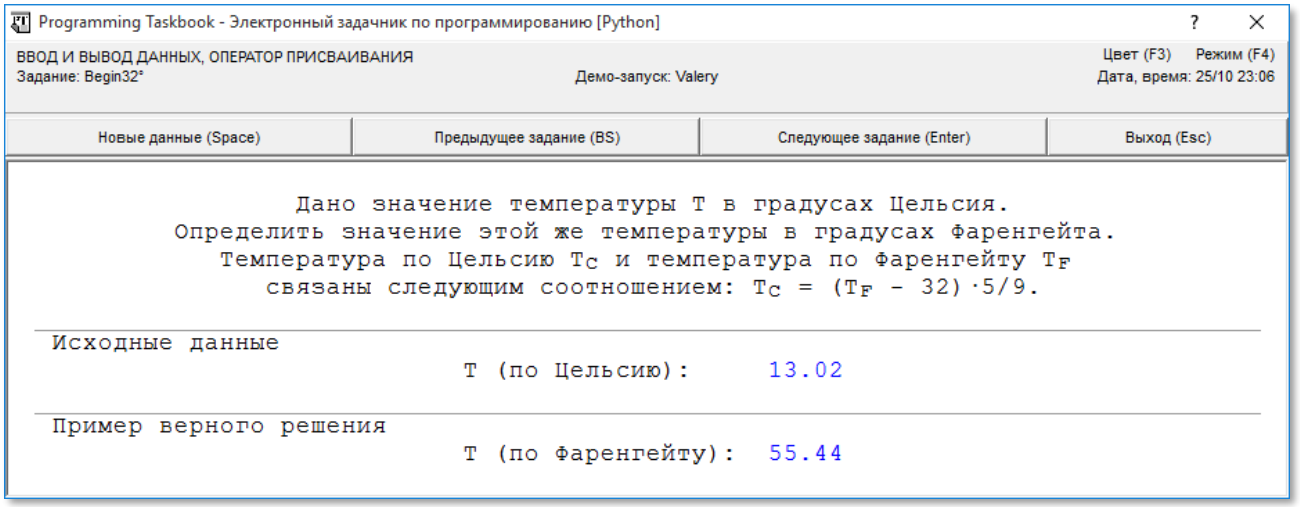


От нас требуется только внимание и аккуратность при переводе математической формулы на язык *Питон*:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin31")

    #температура в градусах Фаренгейта:
    tf = get_float()

    #температура в градусах Цельсия:
    tc = (tf - 32.0) * 5.0 / 9.0
    put(tc)


start(solve)
```

Задание Begin32

Опять задание «наоборот»:



Формула та же самая, что и в предыдущем задании, но теперь нам нужно найти из неё температуру в градусах Фаренгейта.

Начинаем преобразование:

$T_F - 32 = T_C * 9 / 5$

И далее:

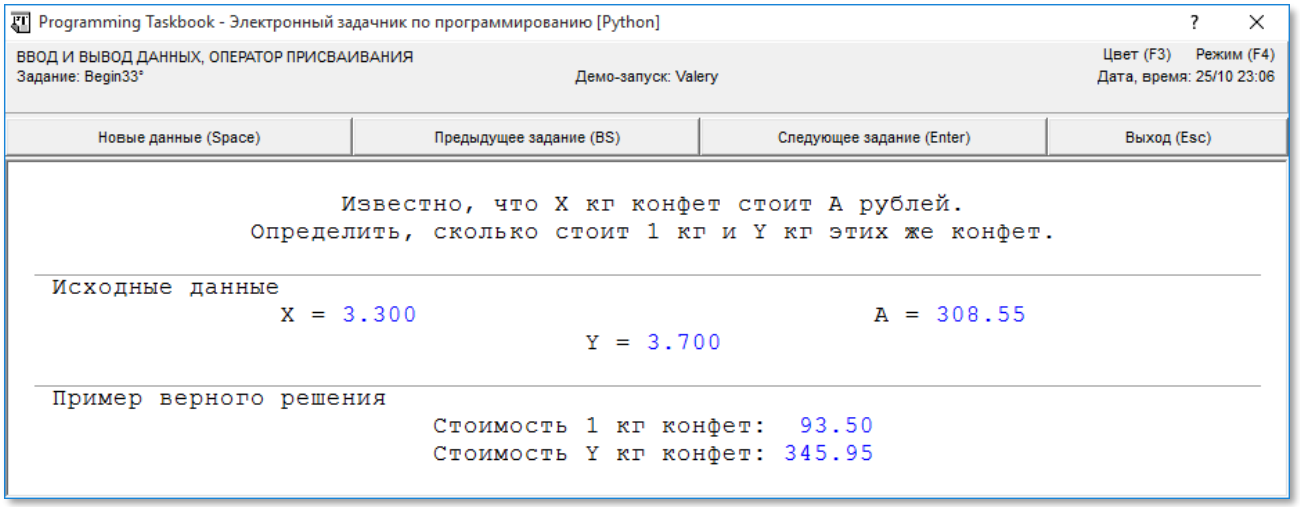
$T_F = T_C * 9 / 5 + 32$

Вот и вся задача!

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin32")

    #температура в градусах Цельсия:
    tc = get_float()

    #температура в градусах Фаренгейта:
    tf = tc * 9.0 / 5.0 + 32.0
    put(tf)


start(solve)
```

Задание Begin33

Очередное задание на простую **пропорцию**:



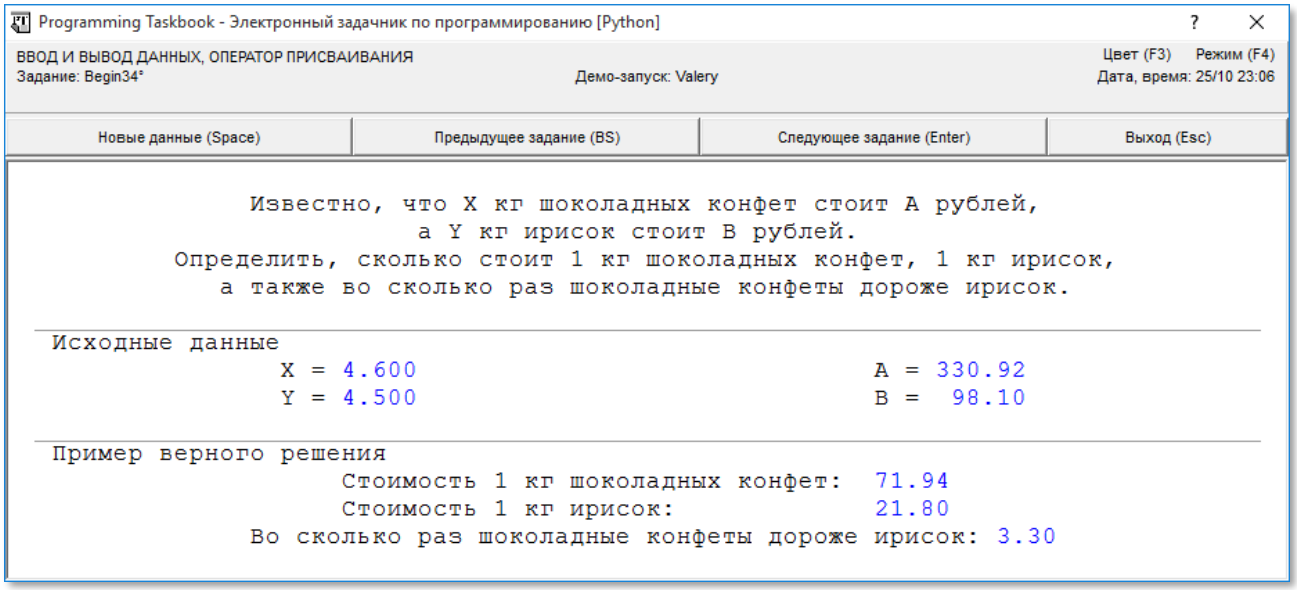
Легко сообразить, что если **X** кг конфет стоят **A** рублей, то **1** кг стоит **A/X** рублей. Если же нужно найти стоимость не 1 кг, а **Y**, то это значение и нужно умножить на **Y**.

То же самое на *Питоне*:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin33")

    #вес и стоимость:
    x = get_float()
    a = get_float()

    #1 кг конфет стоит:
    a1 = a / x
    put(a1)

    #y кг конфет стоят:
    y = get_float()
    ay = a * y / x
    put(ay)


start(solve)
```

Задание Begin34

Опять конфетная задача:



Но уже на 2 сорта конфет.

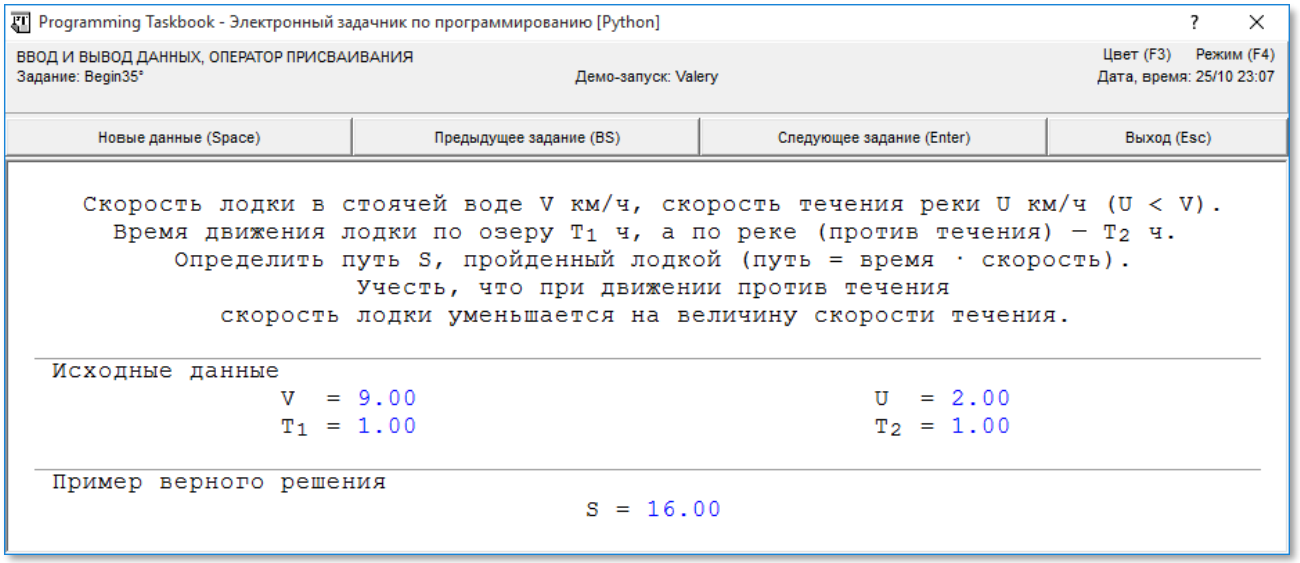
Так как стоимость 1 кг конфет вы уже научились вычислять, то тут и сказочке, то есть задачке – конец:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin34")

    #вес и стоимость шоколадных конфет:
    x = get_float()
    a = get_float()

    #1 кг шоколадных конфет стоит:
    a1 = a / x
    put(a1)

    #вес и стоимость ирисок:
    y = get_float()
    b = get_float()

    #1 кг ирисок стоит:
    b1 = b / y
    put(b1)

    #во ск. раз шок. конфеты дороже ирисок:
```

```
    r = a1 / b1
    put(r)


start(solve)
```

Задание Begin35

Типичная задача из школьной математики:



Так как скорость и время движения нам известны, то **путь** легко вычислить по формуле, данной в условии задачи:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin35")

    #скорости:
    v = get_float()
    u = get_float()

    #время:
    t1 = get_float()
    t2 = get_float()

    #путь лодки по озеру:
    s1 = v * t1
```

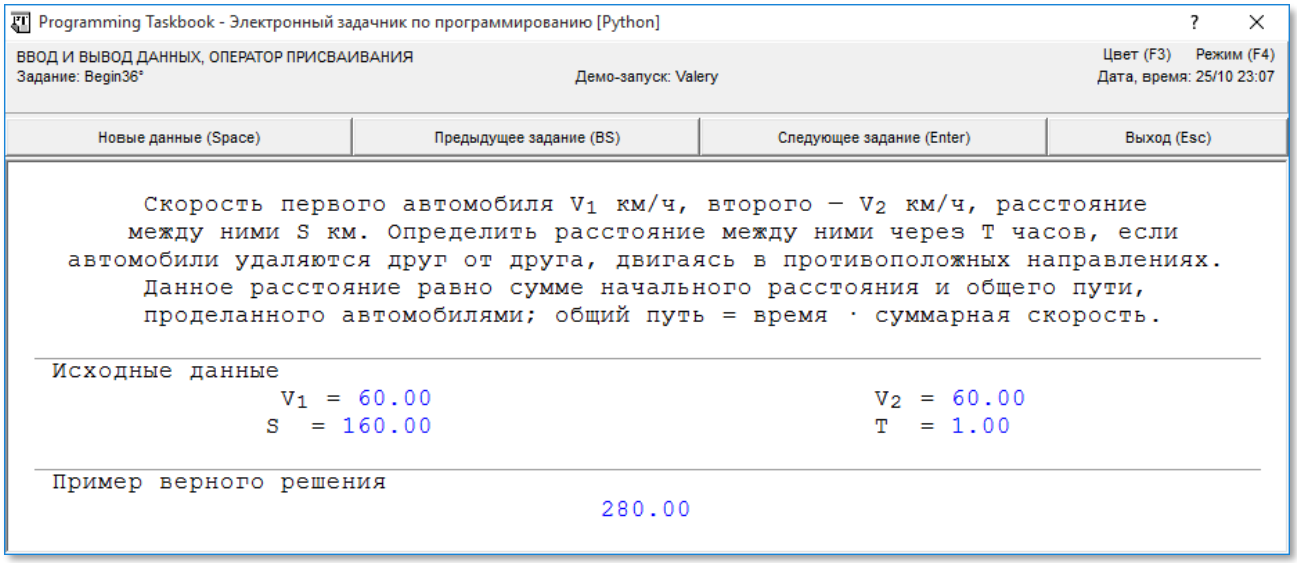
```
    #путь лодки по реке:
    s2 = (v - u) * t2
```

Весь путь равен сумме путей, пройденных лодкой по озеру и по реке:

```
    #общий путь лодки:
    s = s1 + s2
    put(s)


start(solve)
```

Задание Begin36

Также школьная задача:



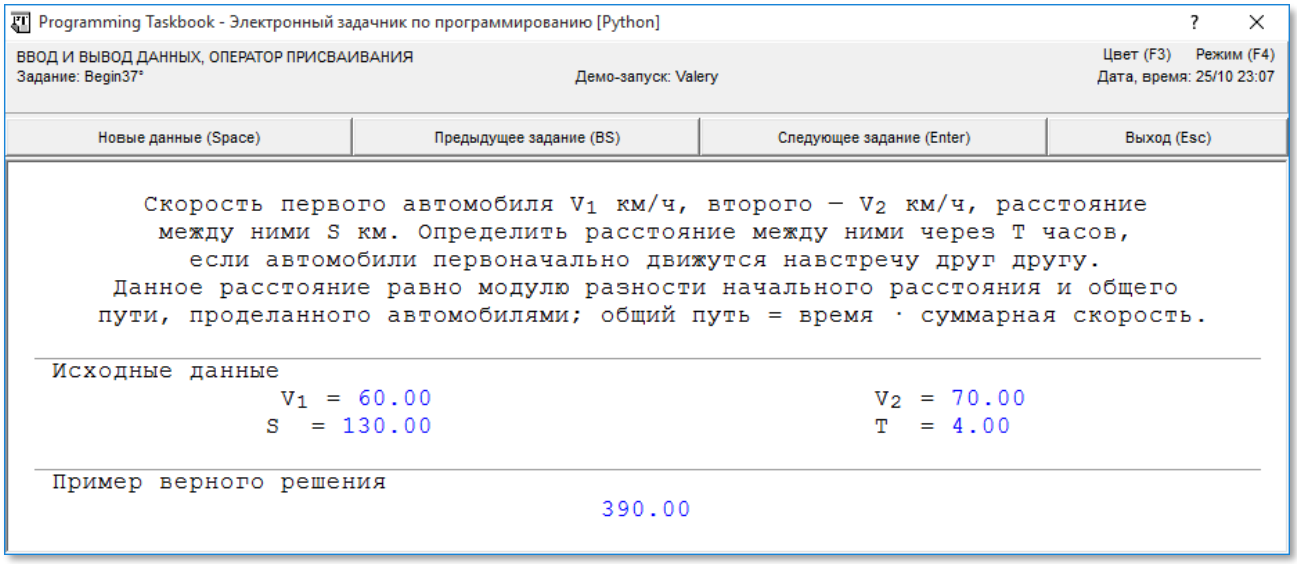
Всё решение уже подробно изложено в условии задачи, поэтому нам остаётся только аккуратно переписать его:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin36")

    #скорости:
    v1 = get_float()
    v2 = get_float()
```

```
    #начальное расстояние между автомобилями:
    s0 = get_float()

    #время движения:
    t = get_float()

    #расстояние между автомобилями через t часов:
    s = s0 + t * (v1 + v2)
    put(s)


start(solve)
```

Задание Begin37

На этот раз автомобили двигаются **навстречу друг другу**, что усложняет нашу и без того простую задачу:



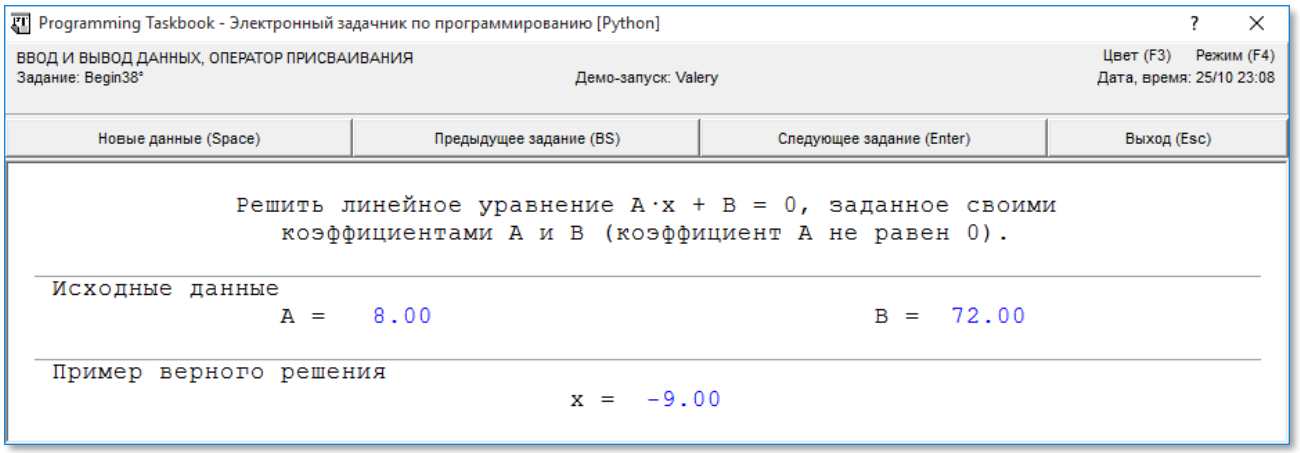
Впрочем, в условии задачи имеется подсказка, которой мы и воспользуемся:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin37")

    #скорости:
    v1 = get_float()
    v2 = get_float()
```

```
    #начальное расстояние между автомобилями:
    s0 = get_float()

    #время движения:
    t = get_float()

    #расстояние между автомобилями через t часов:
    s = abs(s0 - t * (v1 + v2))
    put(s)


start(solve)
```

Задание Begin38

Линейное уравнение – это уравнение с неизвестным(и) в первой степени:



Линейное уравнение в общем виде записывается так:

$$Ax + B = 0$$

Решить уравнение – значит найти *x*.

Для этого нужно преобразовать уравнение так, чтобы в левой части осталось только неизвестное:

$$x = -B / A$$

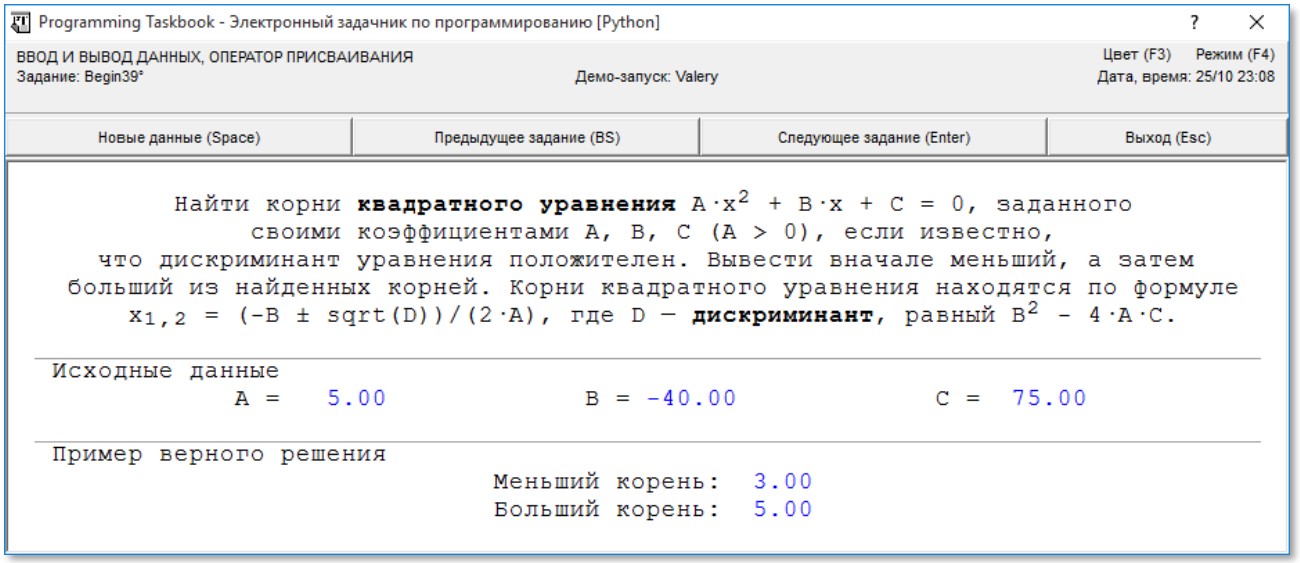
Так как коэффициент **A** не равен нулю, то на него можно делить безбоязненно и безнаказанно.

Переписываем выражение для *x* на *Питон* – и задача решена:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin38")

    #коэффициенты:
    a = get_float()
    b = get_float()

    #решение:
    x = -b / a
    put(x)


start(solve)
```

Задание Begin39

От линейных уравнений переходим к **квадратным**:



К счастью, нам не нужно вспоминать, как решаются квадратные равнения, - вся информация уже содержится в условии задачи.

В первую очередь, нужно вычислить **дискриминант**:

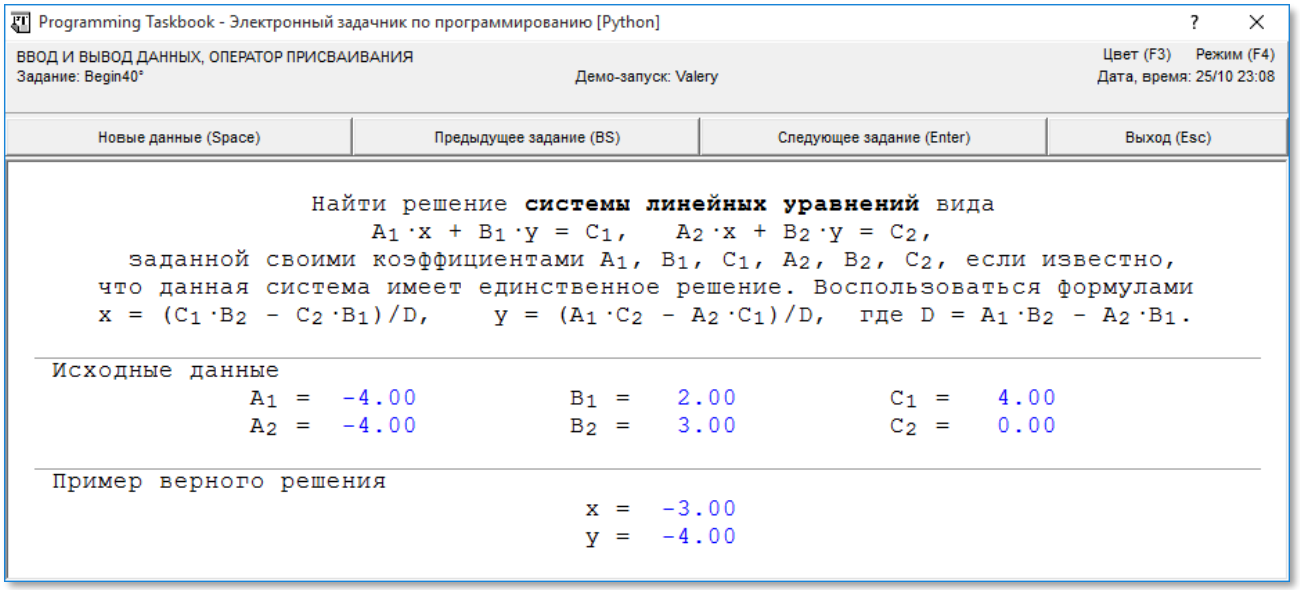
```
# -*- coding: cp1251 -*-
from pt4 import *
import math

def solve():
    task("Begin39")

    #коэффициенты:
    a = get_float()
    b = get_float()
    c = get_float()

    #дискриминант:
    d = b * b - 4 * a * c
```

После чего мы легко находим оба **корня** квадратного уравнения:

```
    #корни уравнения:
    x1 = (-b - math.sqrt(d)) / 2 / a
    x2 = (-b + math.sqrt(d)) / 2 / a
    put(x1, x2)


start(solve)
```

Задание Begin40

Последнее задание в этом наборе:



Решать задачи на систему линейных уравнений очень интересно, но в этом задании нам нужно только переписать информацию из условия задачи:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Begin40")

    #коэффициенты уравнения 1:
    a1 = get_float()
    b1 = get_float()
    c1 = get_float()

    #коэффициенты уравнения 2:
    a2 = get_float()
    b2 = get_float()
    c2 = get_float()

    d = a1 * b2 - a2 * b1

    #решение:
    x = (c1 * b2 - c2 * b1) / d
    y = (a1 * c2 - a2 * c1) / d
    put(x, y)


start(solve)
```

Набор заданий *Integer*

Вы уже освоили технологию выбора заданий, поэтому я только вкратце напомню самые важные действия.

Запустите программу **PT4Load**. Откроется одноимённое диалоговое окно (Рис. 1).

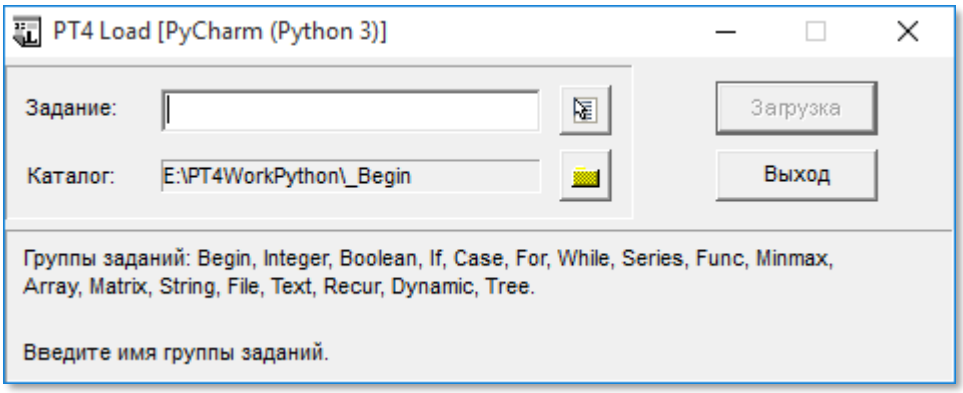


Рис. 1. Смотрим в окно

Внизу вы можете прочитать, какие имеются группы заданий. Нам нужна группа **Integer**.

Набираем в текстовом поле *Задание* буквы *In* (в любом регистре) и получаем внизу диалогового окна полное название группы (Рис. 2).

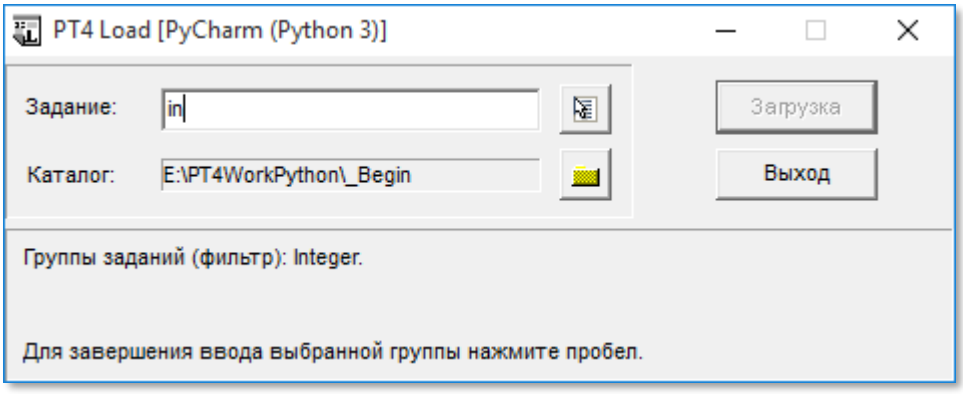


Рис. 2. Фильтруем группы

Достаточно нажать клавишу *ПРОБЕЛ*, чтобы название группы было закончено автоматически. Также вы получите подсказку: в наборе 30 заданий (Рис. 3).

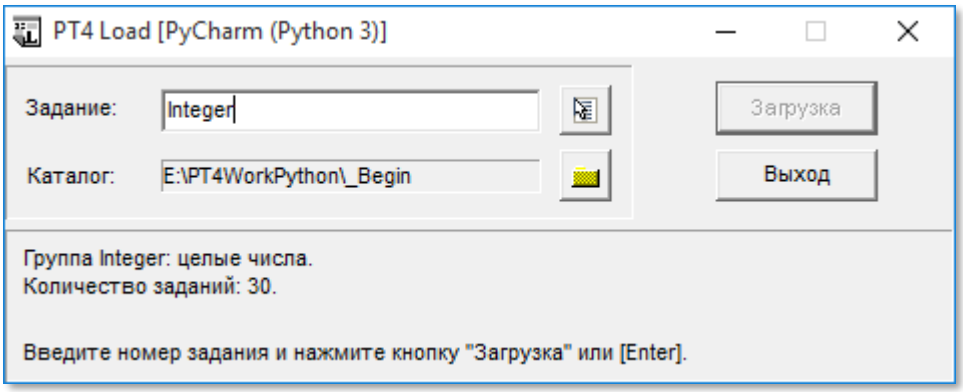


Рис. 3. Полная информация

Начинать нужно с первого задания, поэтому печатаем цифру **1** (Рис. 4).

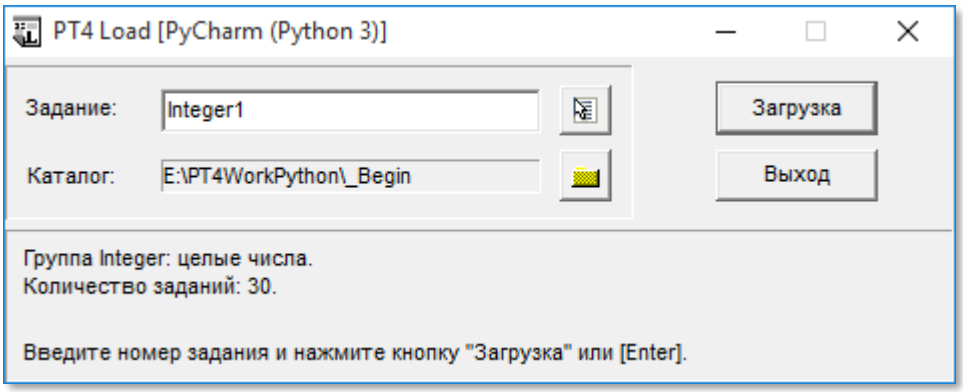


Рис. 4. Пора начинать

Если вы хотите изменить место хранения своих проектов, то нажмите кнопку с жёлтой папкой и укажите новый путь (Рис. 5).

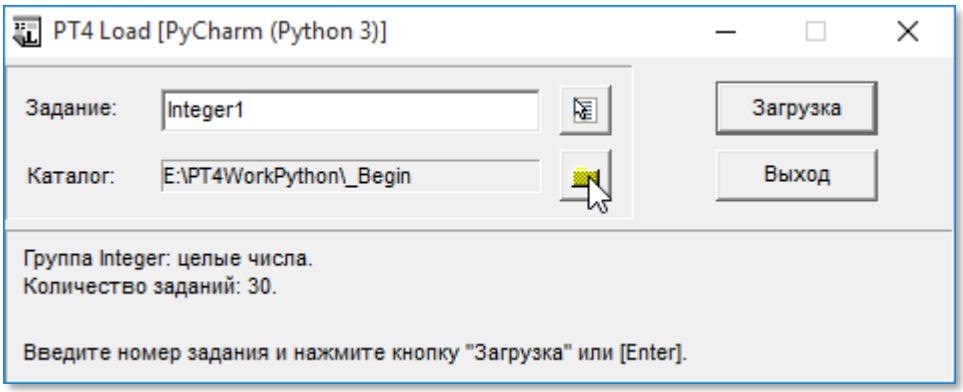


Рис. 5. Пути выбирают

Все приготовления закончены, и мы нажимаем кнопку *Загрузка* или клавишу *ВВОД* (Рис. 6).

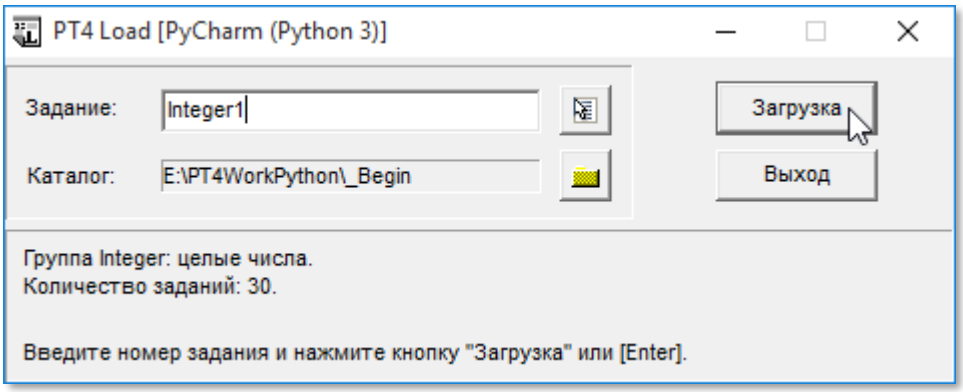


Рис. 6. Всё готово

В *Редакторе кода* появилась страница *Integer1.pyw* с заготовкой кода для задания *Integer1:*

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer1")


start(solve)
```

Итак, **первое задание** такое (Рис. 7).

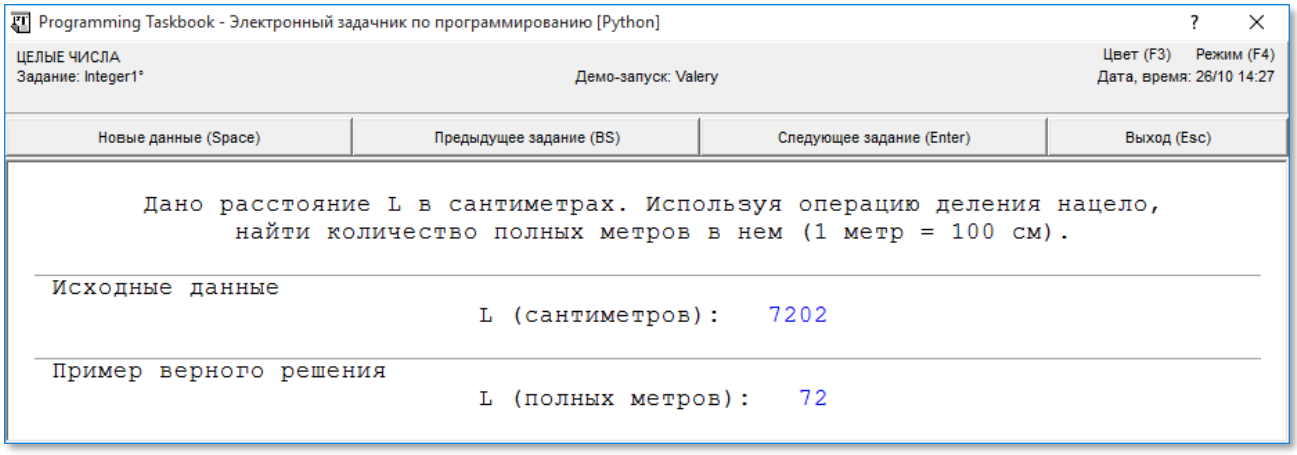


Рис. 7. За работу!

Во всех заданиях должны быть:

- *операторы ввода и вывода get_int и put*

- *арифметические/математические выражения с операциями целочисленного деления // и деления по модулю %*
- *операторы присваивания*

Запомните:

- *Все переменные должны иметь целый тип integer.*
- *Идентификаторы переменных пишутся с маленькой буквы.*
- *Оператор присваивания действует так. Сначала вычисляется значение правой части, а затем оно присваивается переменной в левой части.*
- *В левой части оператора присваивания должна стоять переменная.*
- *В выражениях всегда используется текущее значение переменной.*

Для обозначения **операции целочисленного деления** в *Питоне* используют две косых черты //. **Результат деления имеет целый тип**. Если делимое не делится нацело, то результат (частное) **округляется до ближайшего целого**, которое меньше частного.

Вернёмся к задаче.

В 1 метре содержится 100 сантиметров (если вы забыли, то внимательно прочитайте условие задачи). Это значит, что число сантиметров нужно просто нацело разделить на 100. В результате мы получим **целое** число метров, так как дробная часть частного будет отброшена.

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer1")

    #расстояние в см:
    l = get_int()

    #метраж:
    m = l // 100
    put(m)


start(solve)
```

Для самопроверки и самоконтроля запускаем программу и убеждаемся, что задача решена верно (Рис. 8).

Действительно, если разделить 256 на 100, то получится 2,56. После отбрасывания дробной части останется 2, что и требовалось доказать.

Обращаю ваше внимание, что после отбрасывания дробной части получается не вещественное число 2.0, а **целое** – 2.

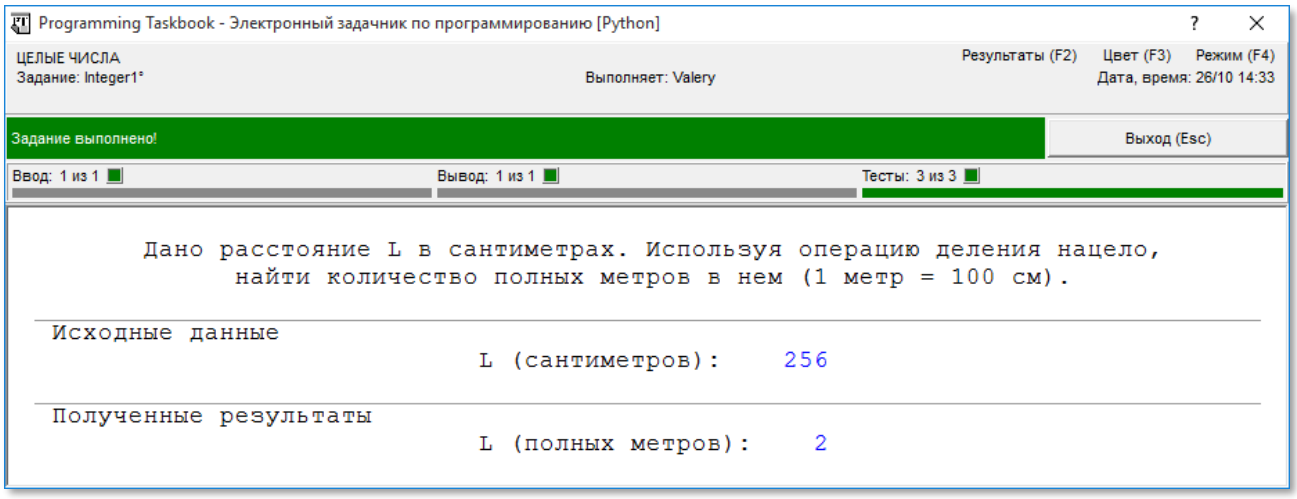
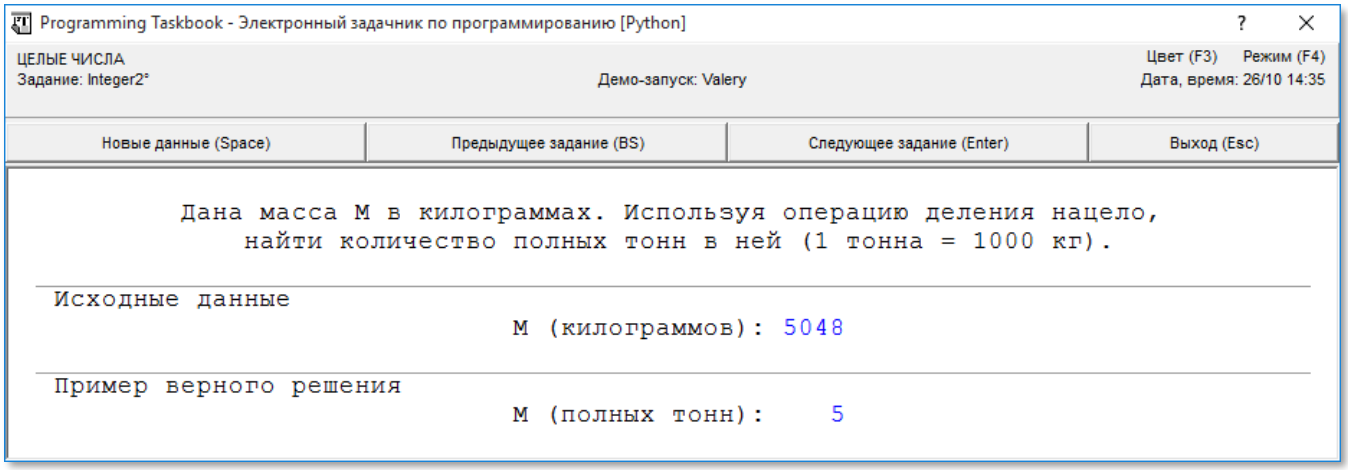


Рис. 10. Прошли все тесты

Задание *Integer2*

Переходим ко второму заданию:

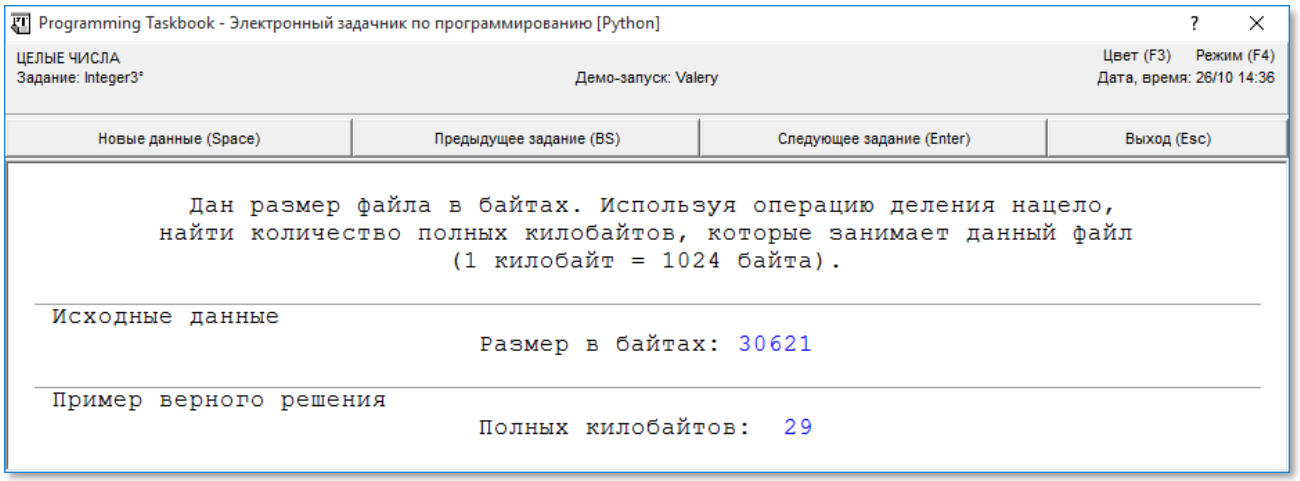


Практически такое же задание, что и первое, только для превращения килограмм в тонны нужно делить на 1000:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer2")

    #масса в кг:
    m = get_int()

    #тоннаж:
    t = m // 1000
    put(t)


start(solve)
```

Задание Integer3

Третье задание воплощает поговорку *повторение – мать учения*:



У программистов кило чего-то это не 1000, как у обычных людей (см. предыдущую задачу), поэтому в килобайте 1024 байтов.

Для нас было бы проще делить на 1000, а компьютеру наоборот – на 1024:

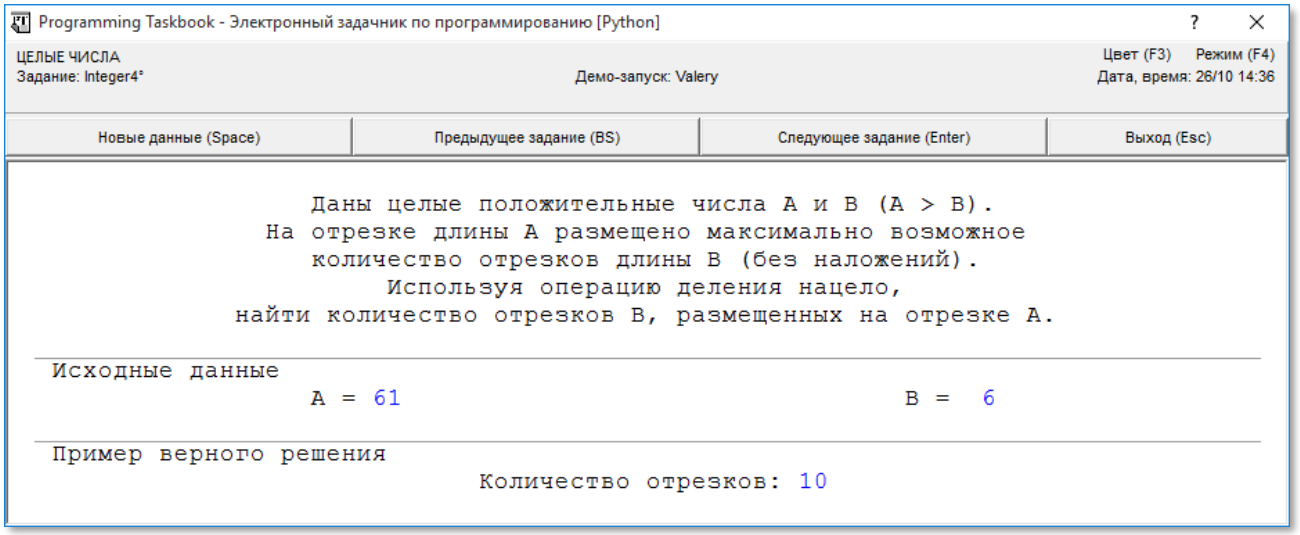
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer3")

    #размер в байтах:
```

```
    b = get_int()

    #размер в килобайтах:
    kb = b // 1024
    put(kb)


start(solve)
```

Если вы когда-нибудь покупали жёсткие, полужёсткие и мягкие диски, то могли убедиться, что изготовители и продавцы дружно считают, что кило это 1000, поэтому объём памяти дисков на самом деле всегда оказывается меньше, чем утверждают эти деятели.

Задание Integer4

Условие четвёртого задания длинное и путанное:



А суть его такова: *поделить число A нацело на число B*.

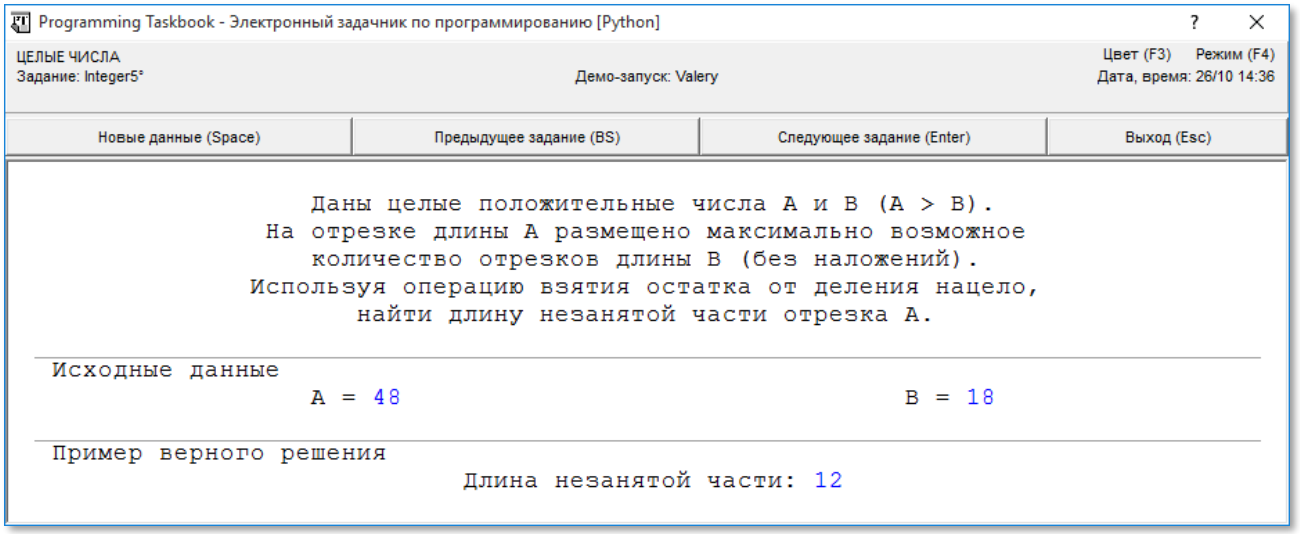
Что мы и делаем с удовольствием:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer4")
```

```
    #числа:
    a = get_int()
    b = get_int()

    #кратность:
    k = a // b
    put(k)


start(solve)
```

Задание Integer5

Добавок к предыдущему заданию:



Эту задачу вполне можно решить, используя только операцию целочисленного деления:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer5")

    #числа:
    a = get_int()
    b = get_int()

    #кратность:
    k = a // b
```

```
    #занято на a:
    z = k * b

    #не занято на a:
    nz = a - z
    put(nz)


start(solve)
```

Но в условии задачи ясно и недвусмысленно требуется использовать **операцию деления по модулю**, которая обозначается символом %. Результат операции – целое число - остаток от деления первого операнда на второй.

Как вы уже знаете, операцию деления по модулю можно заменить операцией целочисленного деления, используя равенство:

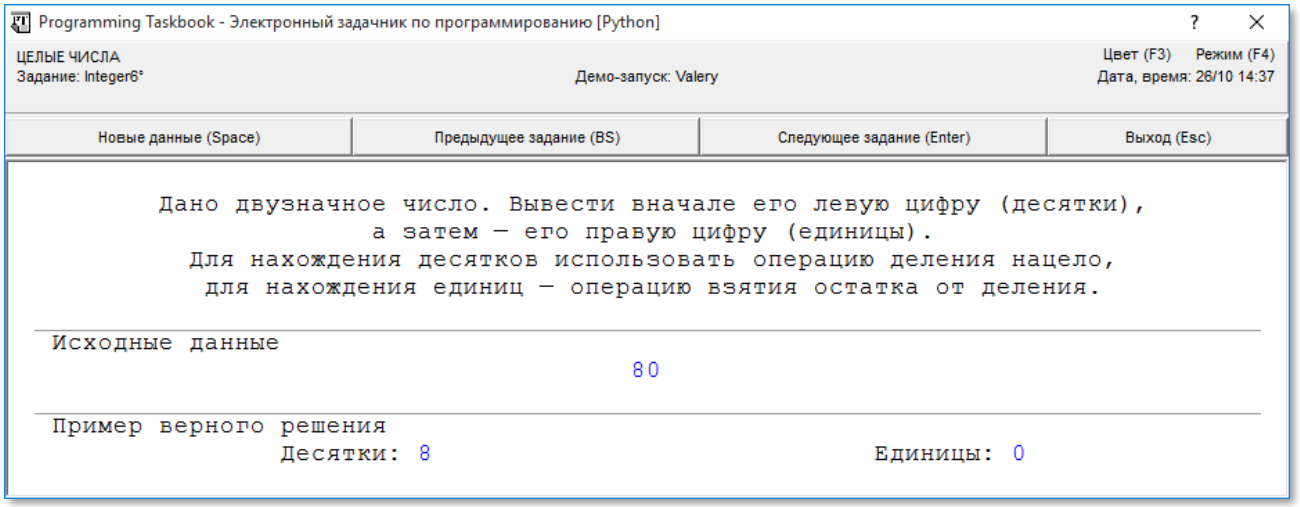
```
x % y = x - (x // y) * y
```

С операцией деления по модулю исходный код программы становится совсем простым:

```
def solve2():
    task("Integer5")

    #числа:
    a = get_int()
    b = get_int()

    #остаток:
    k = a % b
    put(k)


start(solve2)
```

Задание Integer6

Переходим к следующему заданию:

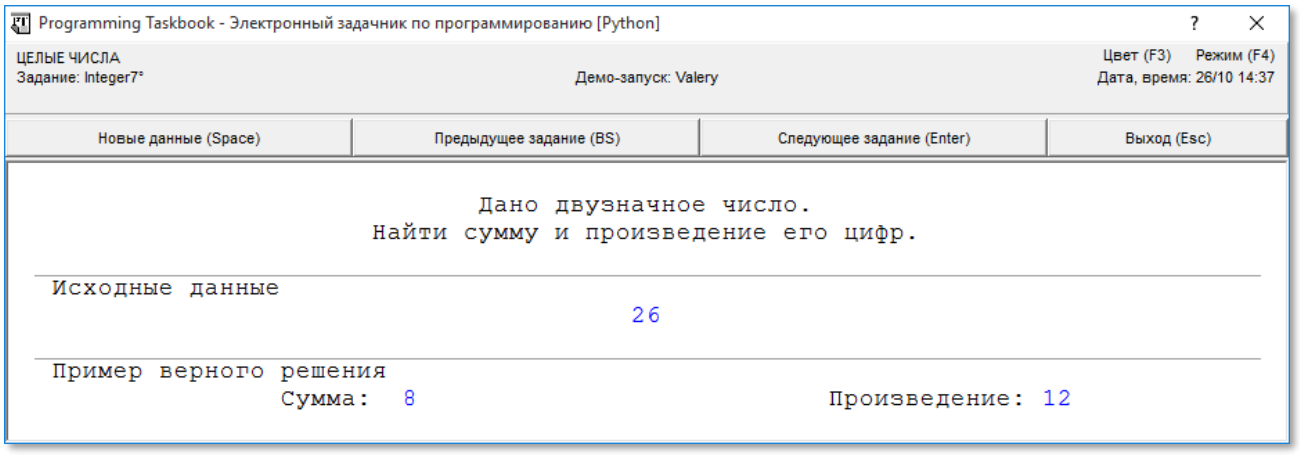


Наконец-то мы получили задание, в котором можно было бы хоть немного подумать, но в условии задачи прямо указано, что число десятков нужно найти с помощью операции деления нацело, а единиц – с помощью операции деления по модулю:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer6")

    #двузначное число:
    de = get_int()

    #число десятков:
    d = de // 10
    #число единиц:
    e = de % 10
    put(d, e)


start(solve)
```

Задание Integer7

Усложнённый вариант шестого задания:

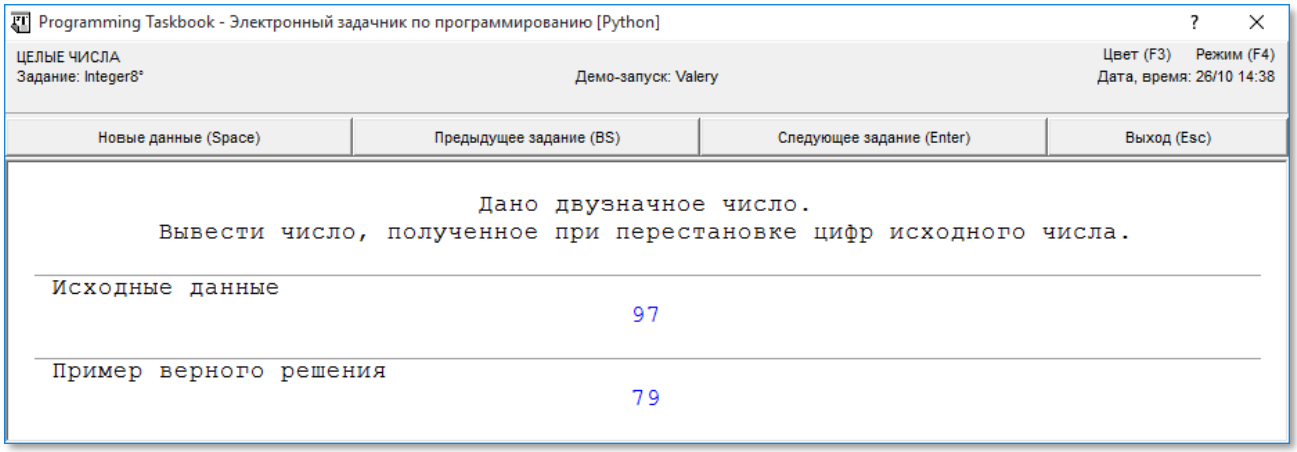


Так как находить цифры двузначного числа вы уже умеете, то вычислить их сумму и произведение не составит большого труда:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer7")

    #двузначное число:
    de = get_int()

    #число десятков:
    d = de // 10
    #число единиц:
    e = de % 10

    #сумма цифр:
    sum = d + e
    #произведение цифр:
    mul = d * e
    put(sum, mul)


start(solve)
```

Задание Integer8

Очередное усложнение:



Чтобы переставить цифры в числе, нужно сначала их найти. Вы это умеете.

При перестановке цифр десятки становятся единицами, а единицы – десятками. Из этого следует, что «перестановленное» число равно:

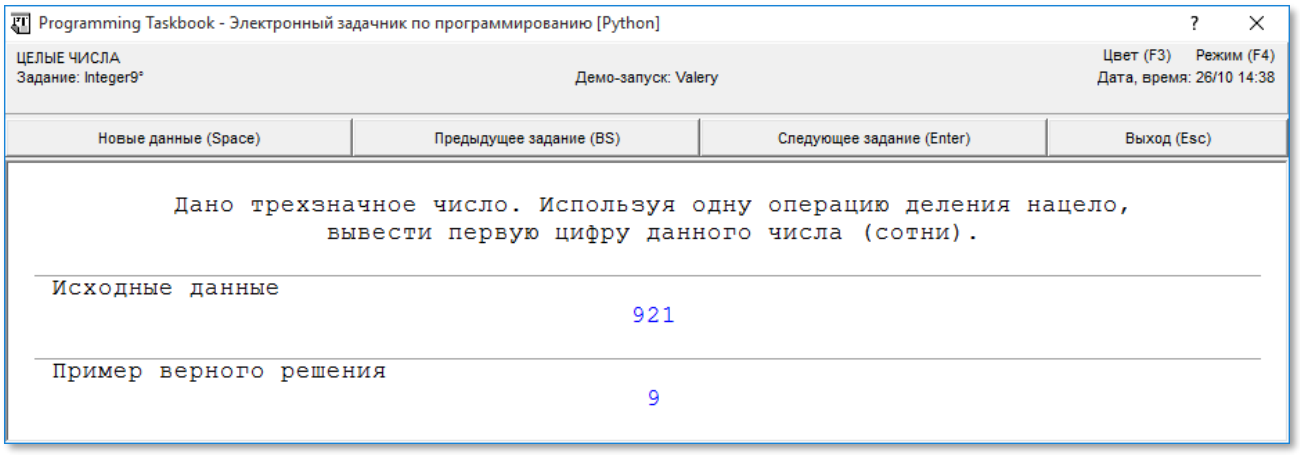
10*единицы + десятки

Вот и вся задача!

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer8")

    #двузначное число:
    de = get_int()

    #число десятков:
    d = de // 10
    #число единиц:
    e = de % 10

    #переставляем цифры:
    ed = e*10 + d
    put(ed)


start(solve)
```

Задание Integer9

Добрались до трёхзначных чисел:

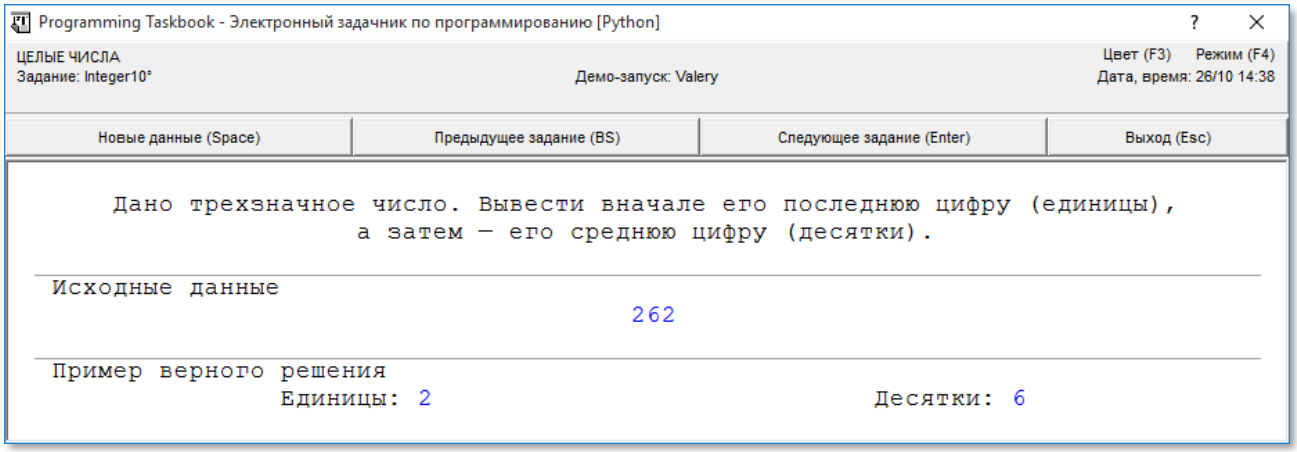


Так как нам нужно найти число сотен в заданном числе, то и делить нужно на сотню:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer9")

    #трёхзначное числр:
    sde = get_int()

    #число сотен:
    s = sde // 100
    put(s)


start(solve)
```

Задание Integer10

На этот раз нам нужно найти число единиц и десятков в трёхзначном числе:



Извлекать число единиц из любого числа вы умеете. Для этого используется операция деления по модулю 10.

Чтобы получить число десятков, нужно… превратить их в единицы, а затем действовать так, как указано выше.

Нетрудно догадаться, что при делении трёхзначного числа на 10 от него останутся две первые цифры. Это значит, что десятки переместятся на место единиц.

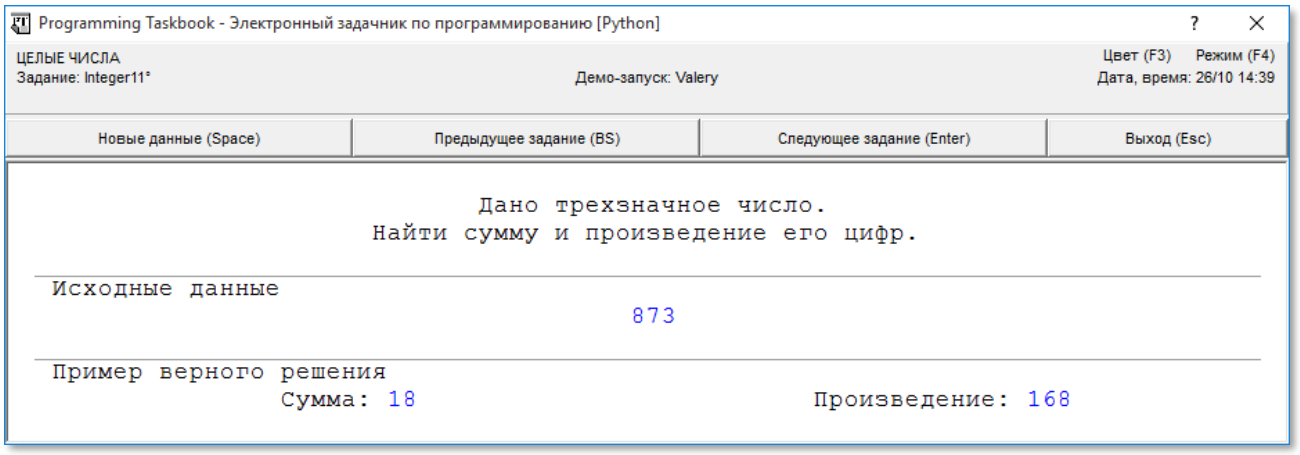
Для сокращения кода весь процесс извлечения десятков можно записать в одну строку:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer10")

    #трёхзначное число:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10
    put(e, d)
```

```
start(solve)
```

Задание Integer11

К десяткам и единицам добавились сотни трёхзначного числа:



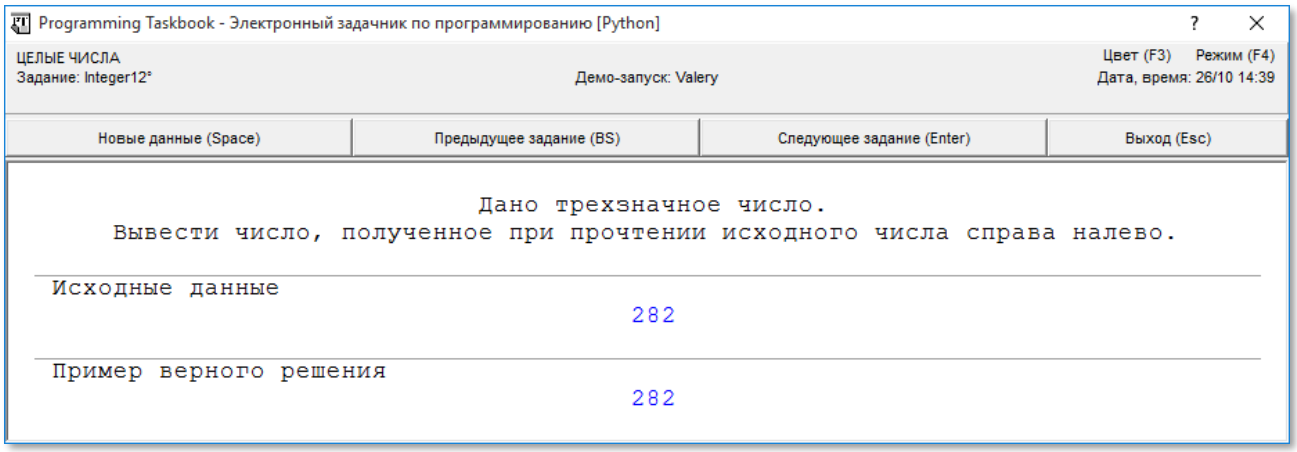
Все эти цифры вы уже умеете извлекать из числа :

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer11")

    #трёхзначное число:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10

    #число сотен:
    s = sde // 100

    #сумма цифр:
    sum = s + d + e
    #произведение цифр:
    mul = s * d * e
    put(sum, mul)
```

```
start(solve)
```

Задание Integer12

Более сложная задача на перестановку цифр в числе:



Все цифры трёхзначного числа мы легко найдём.

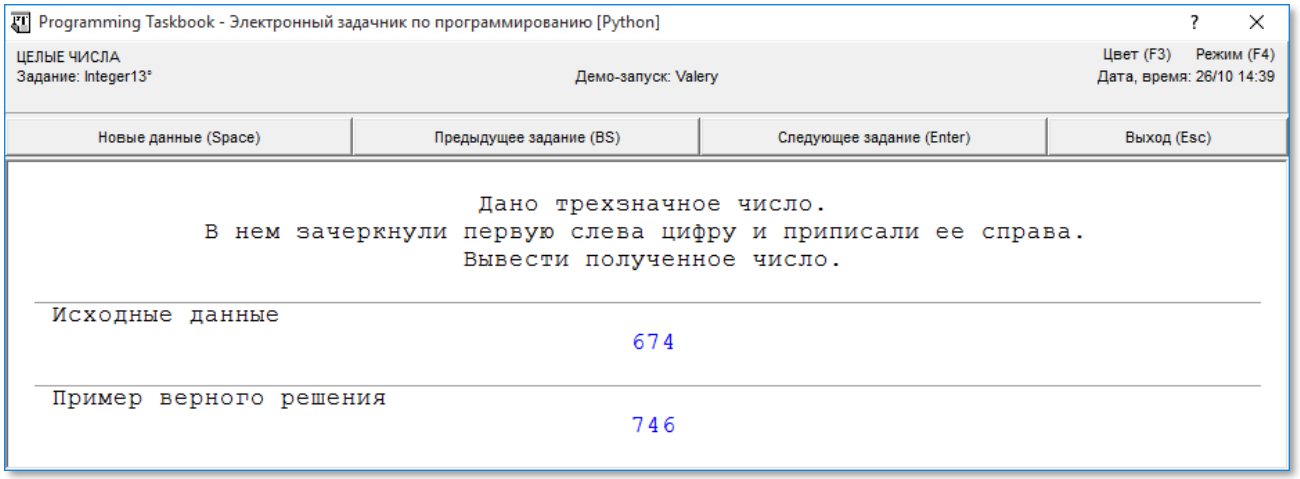
При их перестановке единицы займут место сотен, сотни – место единиц, а десятки так десятками и останутся:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer12")

    #трёхзначное числр:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10

    #число сотен:
    s = sde // 100

    #число-палиндромоид:
```

```
    pal = e * 100 + d * 10 + s
    put(pal)


start(solve)
```

Задание Integer13

Почти такое же задание, что и предыдущее, но цифры переставляются на новый лад:



Чтобы понять, какие места займут цифры в новом числе, внимательно посмотрите на картинку выше.

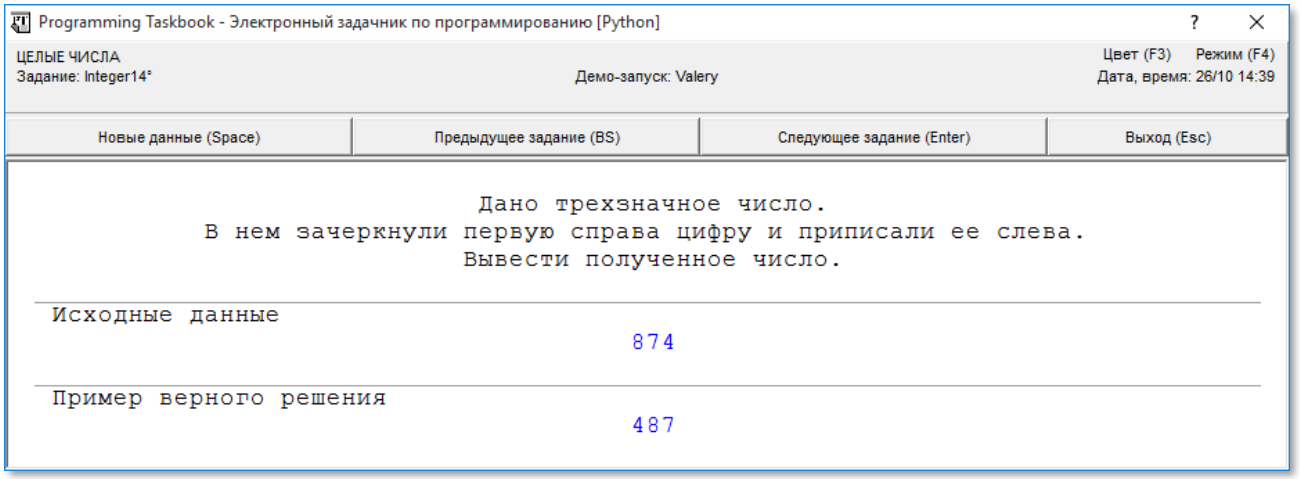
Ага! Сотни становятся единицами, десятки – сотнями, единицы – десятками. Если вы ничего не перепутали, то должен получиться вот такой код:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer13")

    #трёхзначное числр:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10
```

```
    #число сотен:
    s = sde // 100

    #новое число:
    num = d * 100 + e * 10 + s
    put(num)


start(solve)
```

Задание Integer14

Опять задача на потерю бдительности:



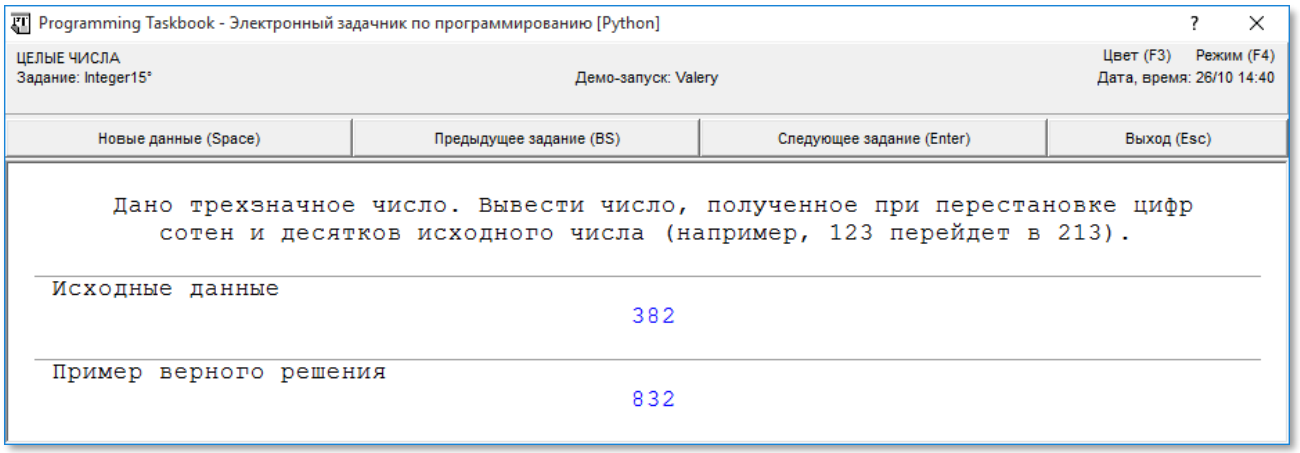
Но если вы её не потеряли, а сберегли, то должны написать такой код:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer14")

    #трёхзначное числр:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10

    #число сотен:
```

```
    s = sde // 100

    #новое число:
    num = e * 100 + s * 10 + d
    put(num)


start(solve)
```

Задание *Integer15*

Как говорится, найдите разницу:



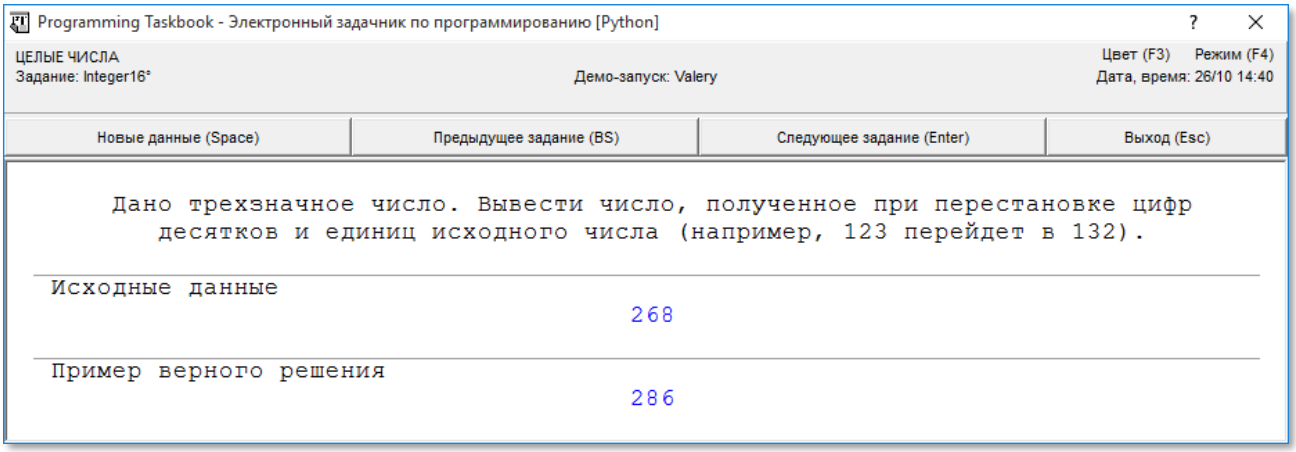
Нашли? – Тогда я вас поздравляю: с **новым кодом**!

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer15")

    #трёхзначное числр:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10

    #число сотен:
    s = sde // 100

    #новое число:
```

```
    num = d * 100 + s * 10 + e
    put(num)


start(solve)
```

Задание Integer16

Опять перестановка цифр:



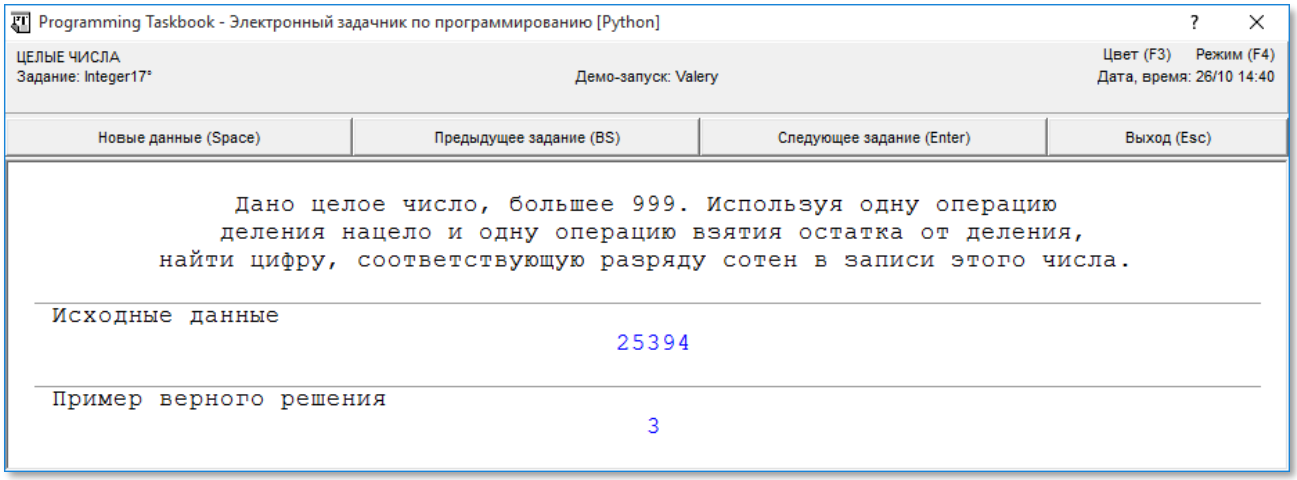
Исходный код в дополнительных комментариях не нуждается:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer16")

    #трёхзначное числр:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10

    #число сотен:
    s = sde // 100

    #новое число:
    num = s * 100 + e * 10 + d
    put(num)
```

```
start(solve)
```

Задание Integer17

Наконец-то пошли большие числа:



Вы уже догадались, что из *n*-значного числа нужно получить 3-значное или я первый?

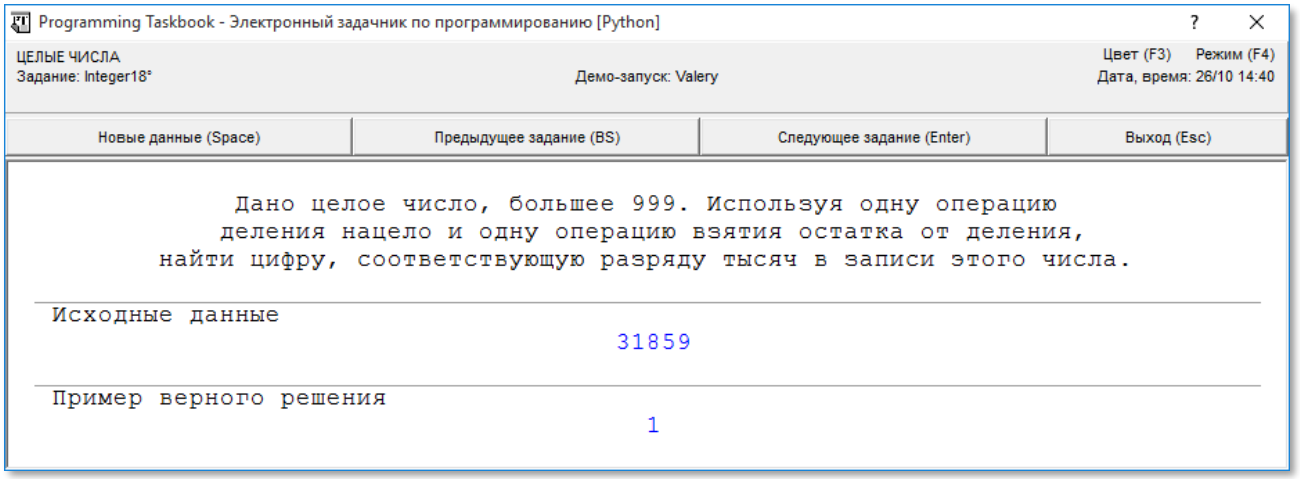
Чтобы выделить из любого числа 3 последние цифры, нужно найти остаток от деления этого числа на 1000:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer17")

    #число:
    num = get_int()

    #трёхзначное число:
    sde = num % 1000

    #число сотен:
    s = sde // 100
    put(s)
```

```
start(solve)
```

Задание Integer18

Числа растут, как грибы после дождя:

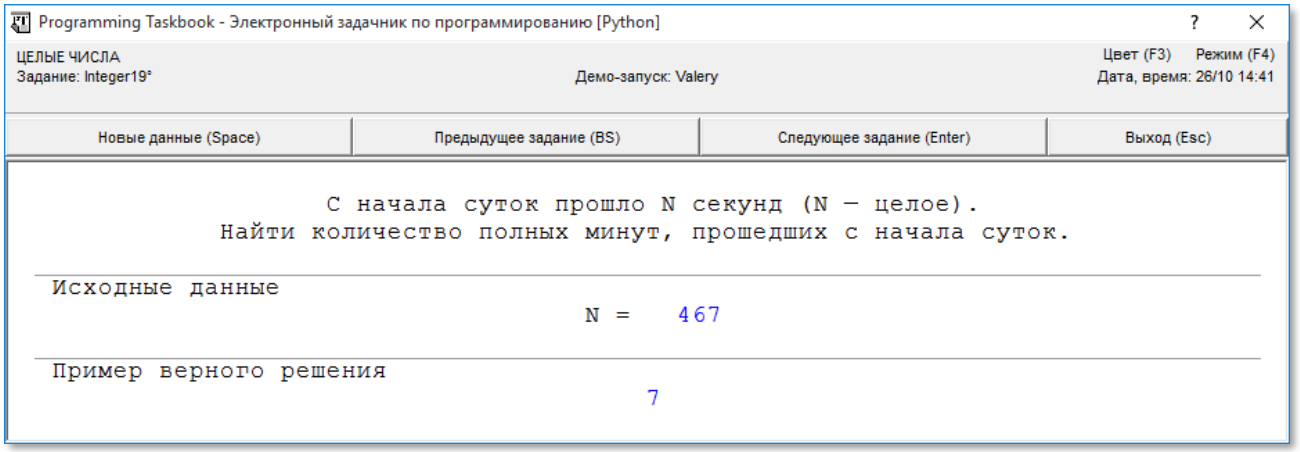


Следуя логике предыдущего задания, мы должны сначала получить 4-значное число, а затем найти в нём число тысяч делением на 1000:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer18")

    #число:
    num = get_int()

    #четырёхзначное число:
    tsde = num % 10000

    #число тысяч:
    t = tsde // 1000
    put(t)


start(solve)
```

Задание Integer19

Возвращаемся к простым задачам на целочисленное деление:

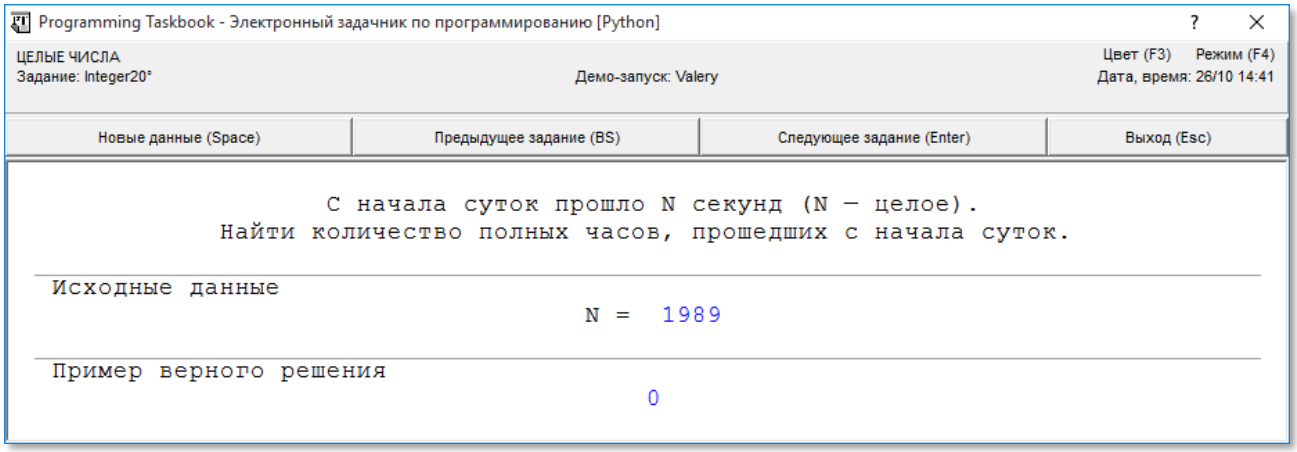


Достаточно вспомнить, что в 1 минуте 60 секунд – и задача решена:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer19")

    #число секунд:
    n = get_int()

    #число минут:
    min = n // 60

    put(min)


start(solve)
```

Задание Integer20

Продолжение часовых задач:

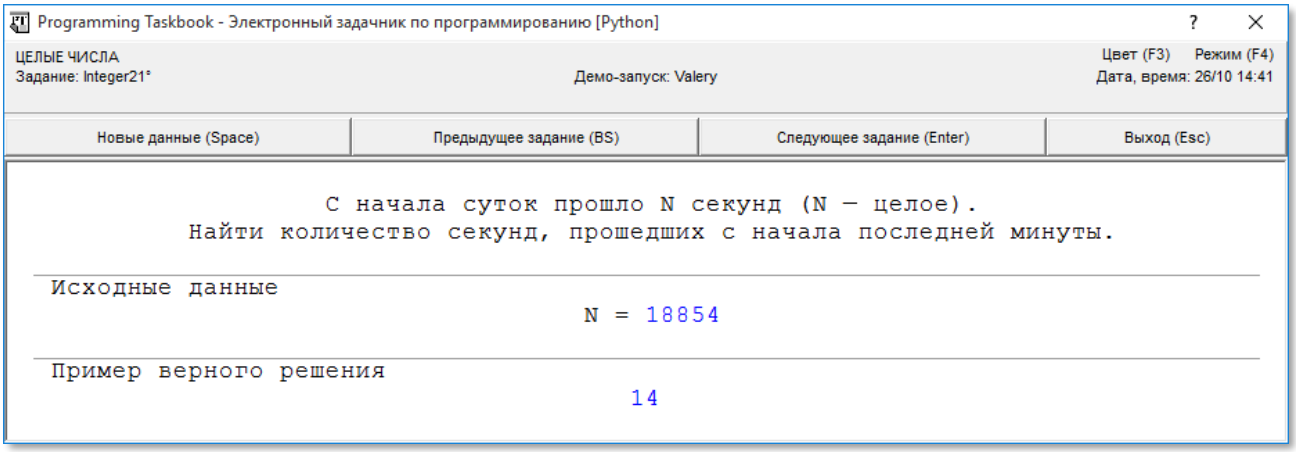


Мы уже вспомнили, что в 1 минуте 60 секунд. Вспоминаем дальше: в 1 часе – 60 минут. Можно найти число часов в одну строчку кода, но обычно программисты это делают не спеша, с расстановкой:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer20")

    #число секунд:
    n = get_int()

    #число минут:
    min = n // 60

    #число часов:
    h = min // 60

    put(h)


start(solve)
```

Задание Integer21

Напряжение растёт:



Прежде чем браться за решение этой задачи, следует в ней разобраться.

До начала последней минуты прошло целое число минут, а осталось меньше одной минуты. Из этого рассуждения следует, что сначала мы должны найти число целых минут в *N* секундах:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer21")

    #число секунд:
    n = get_int()

    #число минут:
    min = n // 60
```

Теперь мы можем узнать, сколько в них секунд:

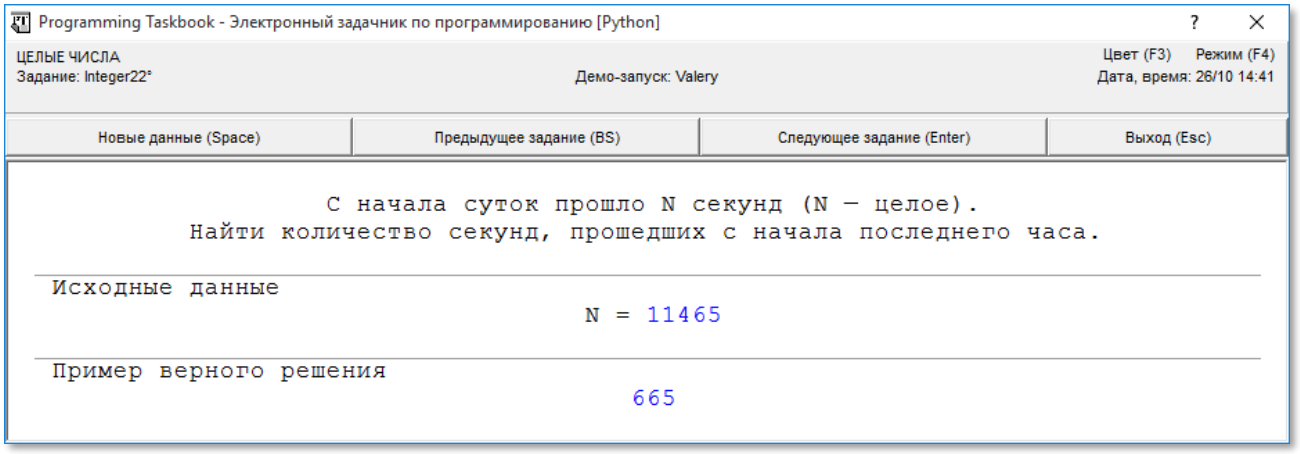
```
#число секунд в min минут:
sec = min * 60
```

Если мы вычтем из общего числа секунд число секунд в полных минутах, то как раз и получим число секунд, прошедших с начала последней минуты:

```
    #остаток секунд:
    ost = n - sec
    put(ost)


start(solve)
```

Задание Integer22

Усугубление предыдущей задачи:



На этот раз мы должны найти число полных часов в *N* секундах:

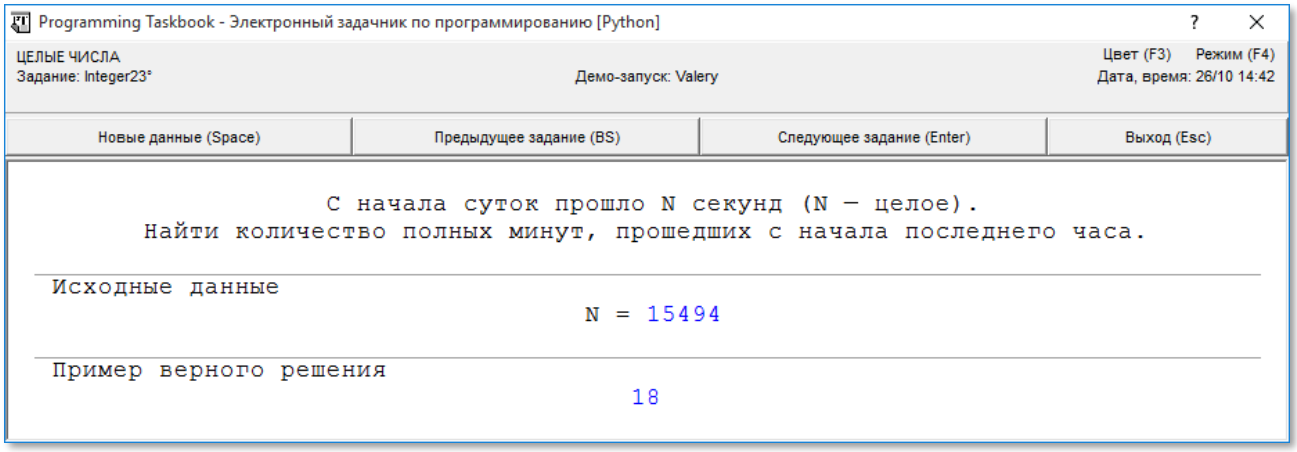
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer22")

    #число секунд:
    n = get_int()

    #число минут:
    min = n // 60

    #число часов:
    h = min // 60
```

Дальше действуем так, как и в предыдущей задаче, учитывая специфику задачи текущей:

```
    #число секунд в h часов:
    sec = h * 60 * 60

    #остаток секунд:
    ost = n - sec
    put(ost)


start(solve)
```

Задание *Integer23*

Минуты сменяют секунды – вот и новая задача:



Находим число полных часов в *N* секундах:

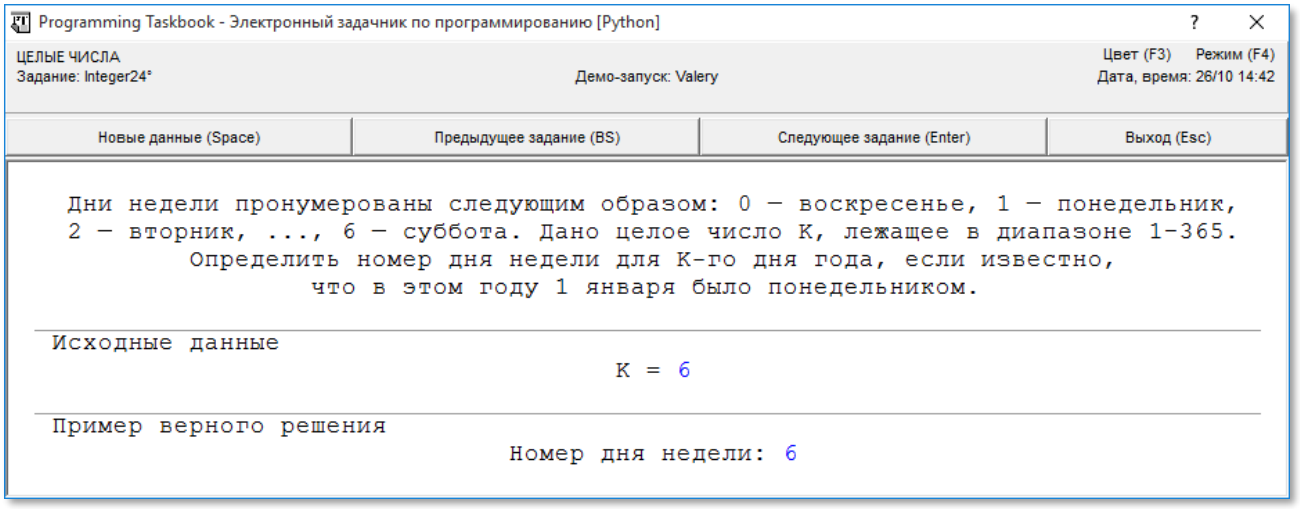
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer23")

    #число секунд:
    n = get_int()

    #число минут:
    min = n // 60

    #число часов:
    h = min // 60
```

А дальше снова – *повторение – мать учения*:

```
    #число минут в h часов:
    minh = h * 60

    #остаток минут:
    ost = min - minh
    put(ost)


start(solve)
```

Задание Integer24

С повышением! От часов мы перешли к календарю:



Если не задаваться вопросом, почему дни пронумерованы таким странным образом, то задача решается в одно действие:

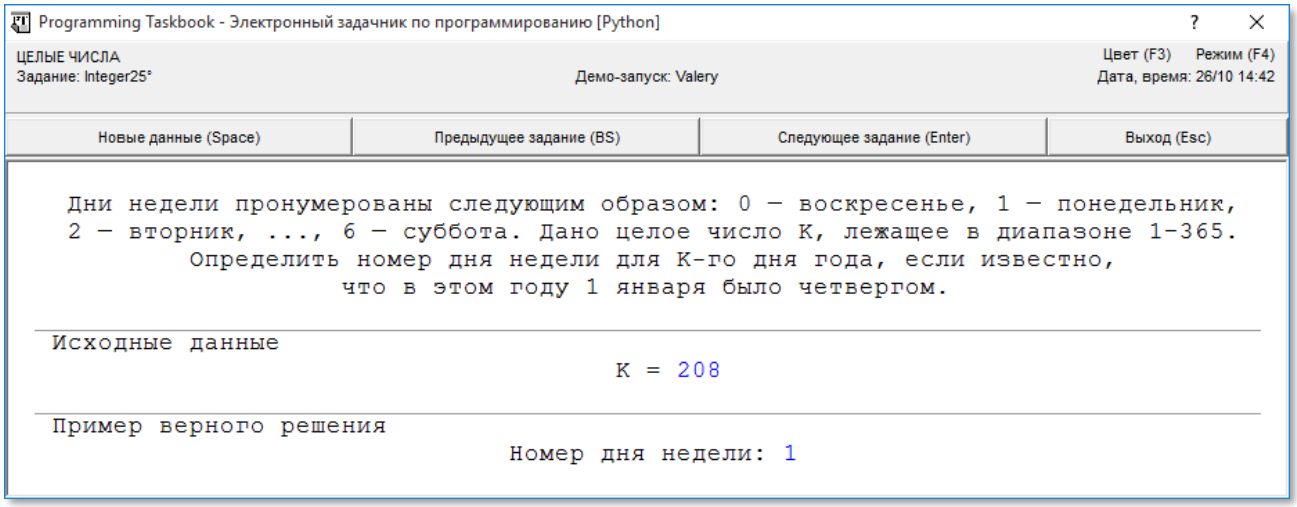
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer24")

    #число:
    k = get_int()

    #номер дня недели:
```

```
    num = k % 7
    put(num)


start(solve)
```

Но это только потому, что первый день года был понедельником!

Задание *Integer25*

Однако не каждый год начинается так удачно:



Следующий год начался сразу с четверга, хотя в жизни так не бывает!

Если первый день года пришёлся на четверг, то для *K = 1* остаток от деления числа прошедших дней на 7 (число дней в неделе) должен равняться *4*, поскольку это был четверг (четвёртый день недели).

Из этой предварительной арифметики следует, что к числу *K* нужно добавить *3*, чтобы получить верный результат:

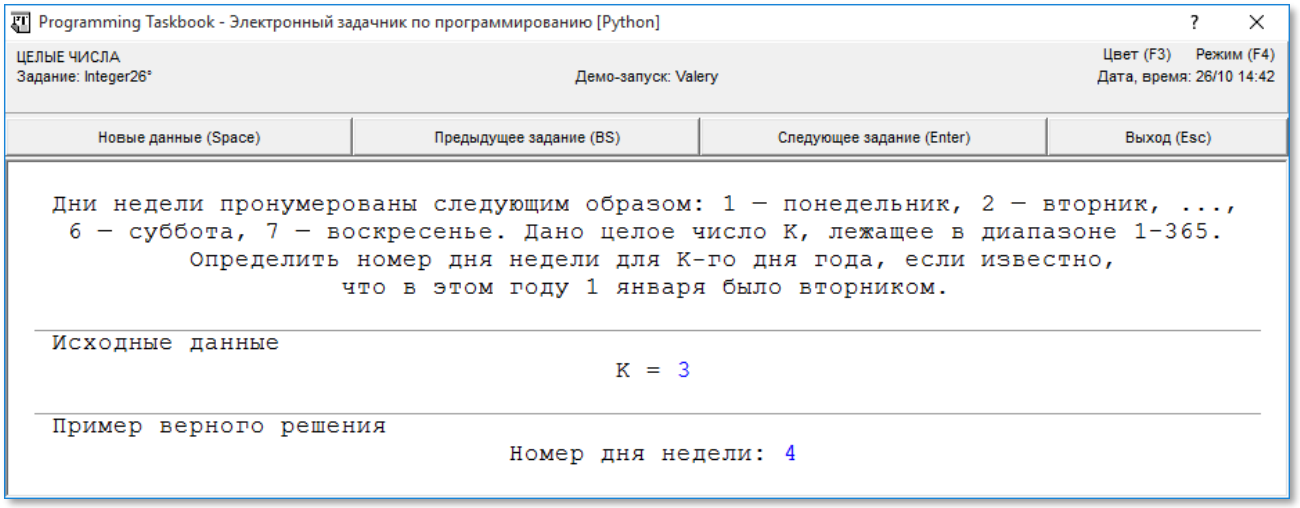
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer25")

    #число:
    k = get_int()

    #номер дня недели:
```

```
    num = (k + 3) % 7
    put(num)


start(solve)
```

Все остальные дни автоматически пронумеруются правильно.

Задание *Integer26*

Путаница с календарём продолжается:



Во-первых, мы должны учесть, что дни теперь нумеруются с единицы, а остаток от деления равен *0..6*. Чтобы получить правильный день, всегда нужно добавлять единицу к остатку от деления.

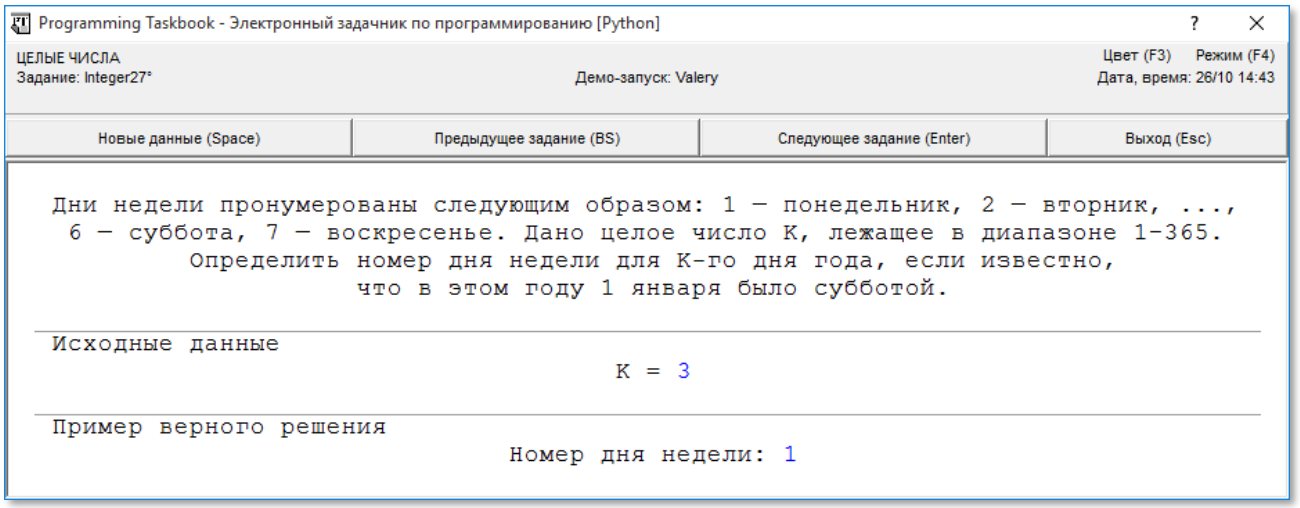
Так как первый день недели был вторником (день номер 2), то при *K = 1* (с учётом добавочной единицы) мы как раз получим двойку, поэтому дни недели вычисляются очень просто:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer26")

    #число:
    k = get_int()

    #номер дня недели:
```

```
    num = k % 7 + 1
    put(num)


start(solve)
```

Задание Integer27

Любите ли вы задачи с календарём так, как их люблю я?



Из предыдущей задачи вы знаете, что к остатку от деления нужно добавить единицу. Это первое.

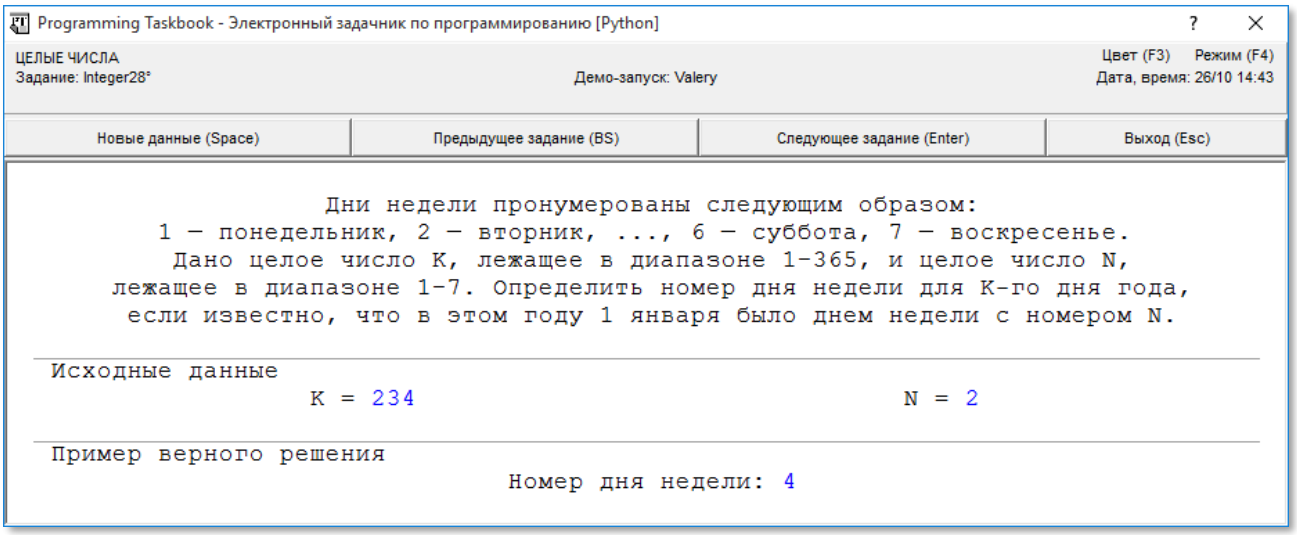
На второе: если бы год начался со вторника, то к числу *K* ничего не нужно было бы добавлять. Но опять случилась оказия, и год начался с субботы. Это значит, что к числу *K* нужно добавить: среда – 1, четверг – 2, пятница – 3, **суббота – 4**. Логично? – Аналогично!

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer27")

    #число:
    k = get_int()

    #номер дня недели:
    num = (k + 4) % 7 + 1
    put(num)
```

```
start(solve)
```

Задание *Integer28*

Как вы помните, на Острове невезения не было календаря. А у нас каждый год – новый календарь:



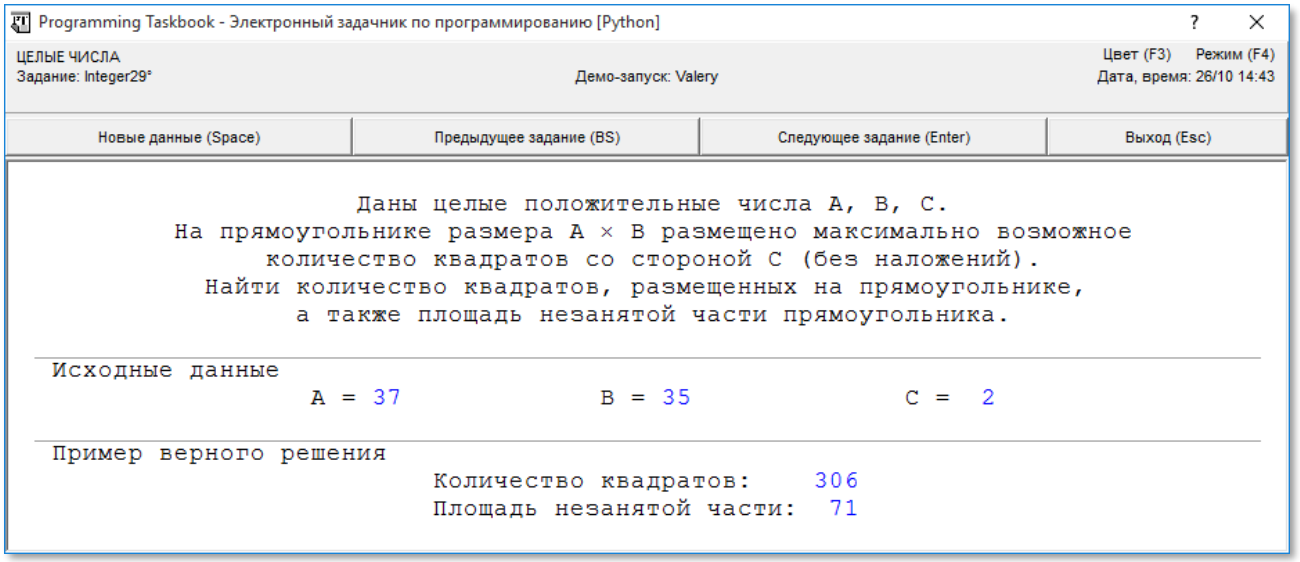
С добавочной единицей всё понятно. А вот, что делать с числом *N*? – Рассуждаем: если **N = 2**, то к числу *K* нужно добавить *0*. Это следует из задачи про вторник. Если **N = 6**, то к числу *K* нужно добавить *4*. А это следует уже из задачи про субботу. Небольшое умственно-мышечное усилие – и вот его результат: из числа *N* необходимо вычесть *2*, чтобы получить правильный день недели по новому календарю:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer28")

    #числа:
    k = get_int()
    n = get_int()

    #номер дня недели:
    num = (k + n - 2) % 7 + 1
    put(num)
```

```
start(solve)
```

Задание Integer29

Резкий и неожиданный переход от календаря к прямоугольникам:



Так как квадраты должны целиком поместиться на предложенной им жилплощади, то мы легко найдём общее число квадратов, удобно расположившихся по ширине и по высоте прямоугольника:

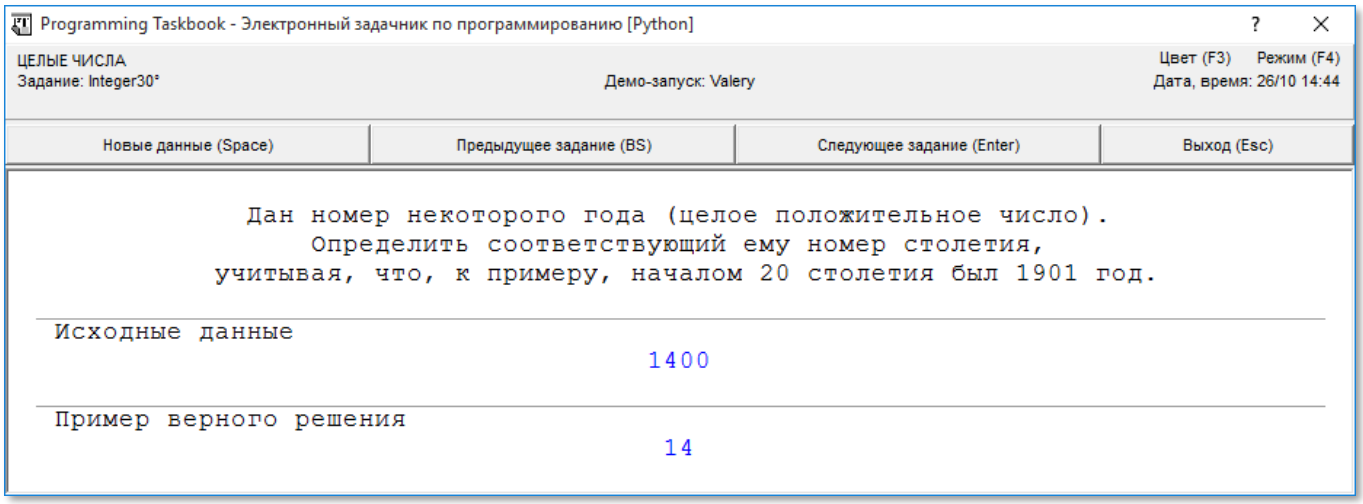
```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer29")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    #число квадратов по ширине:
    numw = a // c
    #число квадратов по высоте:
    numh = b // c

    #общее число квадратов:
    num = numw * numh
```

Если из площади прямоугольника вычесть площадь этих квадратов, то мы получим свободную площадь:

```
    #площадь квадратов:
    s = num * c * c

    #остаток площади:
    ost = a * b - s
    put(num, ost)


start(solve)
```

Задание *Integer30*

Резкий возвратный скачок к календарю:



Номер столетия легко найти делением по модулю 100. Но при этом мы должны учитывать, что отсчёт времени начался не с нулевого года, а с первого, и столетие также было не нулевое, а первое. В таких случаях, как вы уже знаете, к остатку от деления нужно добавлять 1.

А так как первый год был именно первым, то из номера года нужно 1 вычесть:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Integer30")

    #номер года:
    ng = get_int()

    #номер столетия:
    ns = (ng - 1) // 100 + 1
    put(ns)


start(solve)
```

Если вы помните, была большая календарная неразбериха с новым тысячелетием. Гуманитарно образованные граждане считали, что новое тысячелетие начнётся 1 января 2000 года. Математически образованные сограждане, пользуясь таблицей умножения и здравым смыслом, утверждали, что 2 тысячелетия – это 2000 прошедших лет. Следовательно, новое, третье тысячелетие начнётся 1 января 2001 года. Так оно и началось!

Кроме гуманитарно и математически образованных граждан, есть ещё и политики, на календаре у которых до сих пор **мрачное средневековье**.

Ученье - свет!

Набор заданий *Boolean*

Сразу приступаем к решению задач. А в этом наборе их ровно 40 штук.

Задание *Boolean1*

Итак, **первое задание** такое:

Дано целое число A. Проверить истинность высказывания:
«Число A является положительным».

Исходные данные

A = -41

Пример верного решения

False

В этой задаче **высказывание** – это логическое выражение, которое может иметь 2 значения:

- **True** – истинное высказывание
- **False** – ложное высказывание

Высказывание *Число A является положительным* можно записать на языке *Питон* так:

A > 0

На этом решение заканчивается:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean1")

    #число:
    a = get_int()

    #высказывание:
    v = a > 0
    put(v)


start(solve)
```

Не забывайте печатать ответ на экране!

Задание Boolean2

Переходим ко **второму** заданию:

Дано целое число A. Проверить истинность высказывания:
«Число A является нечетным».

Исходные данные

A = -48

Пример верного решения

False

Целое число называется **нечётным**, если оно не делится на 2 без остатка. На языке *Питон* это определение можно записать так:

```
a % 2 != 0
```

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean2")

    #число:
    a = get_int()

    #высказывание:
    v = a % 2 != 0
    put(v)


start(solve)
```

Задание Boolean3

Второе задание, но наоборот:

Дано целое число A. Проверить истинность высказывания:
«Число A является четным».

Исходные данные

A = 26

Пример верного решения

True

Если целое число – **чётное**, то оно делится на 2 без остатка:

a % 2 = 0

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean3")

    #число:
    a = get_int()

    #высказывание:
    v = a % 2 == 0
    put(v)


start(solve)
```

Задание Boolean4

Высказывание уже дано в условии задания:

Даны два целых числа: A, B. Проверить истинность высказывания:
«Справедливы неравенства A > 2 и B ≤ 3».

Исходные данные

A = 2 B = 0

Пример верного решения

False

Нам нужно только заменить союз *и* знаком логической операции *логическое И* – **and**. Отдельные логические выражения, входящие в состав сложного, для наглядности лучше заключать в круглые скобки:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean4")

    #числа:
    a = get_int()
    b = get_int()

    #высказывание:
    v = (a > 2) and (b <= 3)
    put(v)


start(solve)
```

Задание Boolean5

Здесь на смену логической операции *И* пришла другая логическая операция – *ИЛИ* - **or**:

Даны два целых числа: A, B. Проверить истинность высказывания:
«Справедливы неравенства A ≥ 0 или B < -2».

Исходные данные

A = 1 B = -5

Пример верного решения

True

Никаких сложностей при решении этой задачи вы не встретите:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean5")

    #числа:
    a = get_int()
    b = get_int()

    #высказывание:
    v = (a >= 0) or (b < -2)
    put(v)


start(solve)
```

Задание Boolean6

В этом задании сложное высказывание записано так, как принято в математике:

Даны три целых числа: A, B, C. Проверить истинность высказывания:
«Справедливо двойное неравенство A < B < C».

Исходные данные

A = 87 B = 89 C = 67

Пример верного решения

False

В большинстве языков программирования отдельные высказывания должны соединяться логическими операторами **AND** и/или **OR**. Таким образом, сначала нужно разбить сложное высказывание на простые. В данном случае их два:

A < B и B < C

Так как они должны выполняться (быть истинными) **одновременно** (оба), то для объединения простых высказываний нужно использовать операцию *логического* И, то есть **and**:

```
v = (a < b) and (b < c)
```

Но в *Питоне* вполне допустимы и сложные высказывания, поэтому решение задачи оказывается совсем простым:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean6")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    #высказывание:
    #v = (a < b) and (b < c)
    v = a < b < c
    put(v)
```

```
start(solve)
```

Задание Boolean7

Задача на числовую ось:

```
        Даны три целых числа: A, B, C. Проверить истинность высказывания:
                «Число B находится между числами A и C».

Исходные данные
                A = 84              B = 69              C = 25

Пример верного решения
                                    True
```

«Нарисуем» числовую ось и отметим на ней заданные числа:

A---B---C
C---B---A

Первая схема описывает ситуацию, когда *A* – самое маленькое из трёх чисел. **Вторая** – когда самое большое.

Если число *B* находится между A и C, то **в первом случае** должно выполняться двойное неравенство:

A < B < C

Во втором случае неравенство должно быть таким:

C < B < A

Чтобы учесть **оба** случая, нужно соединить эти неравенства логической операцией ИЛИ:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean7")

    #числа:
    a = get_int()
```

```
    b = get_int()
    c = get_int()

    #высказывание:
    #v = ((a < b) and (b < c)) or ((a > b) and (b > c))
    v = (a < b < c) or (c < b < a)
    put(v)


start(solve)
```

Задание Boolean8

Задание на сложное логическое выражение:

Даны два целых числа: A, B. Проверить истинность высказывания:
«Каждое из чисел A и B нечетное».

Исходные данные

A = -7 B = 7

Пример верного решения

True

Нечётные числа вы уже умеете определять, так что нам осталось соединить 2 простых выражения в одно сложное.

Если *каждое из чисел A и B нечётное*, то они нечётные **одновременно**, то есть для связки нужно использовать *логическую операцию И*:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean8")

    #числа:
    a = get_int()
    b = get_int()

    #высказывание:
    v = (a % 2 != 0) and (b % 2 != 0)
    put(v)


start(solve)
```

Задание Boolean9

Вариация предыдущего задания:

Даны два целых числа: A, B. Проверить истинность высказывания:
«Хотя бы одно из чисел A и B нечетное».

Исходные данные

A = -4 B = -1

Пример верного решения

True

Высказывание *Хотя бы одно из чисел нечётное означает*, что:

- *ИЛИ A нечётное*
- *ИЛИ B нечётное*
- *ИЛИ A и B нечётные*

Так как не обязательно оба числа одновременно должны быть нечётными, то для связки простых логических выражений нужно использовать *логическую операцию ИЛИ*:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean9")

    #числа:
    a = get_int()
    b = get_int()

    #высказывание:
    v = (a % 2 != 0) or (b % 2 != 0)
    put(v)


start(solve)
```

Задание Boolean10

А вот это уже интересно:

Даны два целых числа: A, B. Проверить истинность высказывания:
«Ровно одно из чисел A и B нечетное».

Исходные данные

A = 5 B = 3

Пример верного решения

False

Нечётность заданных чисел мы проверяем, как обычно, поэтому простые высказывания будут точно такими же, как и в предыдущих заданиях. Вопрос в том, как из них получить сложное высказывание/выражение.

Здесь самое время вспомнить о логической операции **XOR**, которая в *Питоне* обозначается символом **^**:

> **XOR** - логическая операция **ИСКЛЮЧАЮЩЕЕ ИЛИ**
> Результат операции тогда и только тогда будет истинным, если один операнд истинен, а второй ложен, во всех остальных случаях результат будет ложным.

Из её описания становится понятно, что сложное логическое выражение будет быть истинным только тогда, когда одно из простых выражений истинное, а второе – ложное. Следовательно, для связки простых логических выражений нужно использовать операцию **ИСКЛЮЧАЮЩЕЕ ИЛИ**:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean10")

    #числа:
    a = get_int()
    b = get_int()

    #высказывание:
    v = (a % 2 != 0) ^ (b % 2 != 0)
    put(v)
```

```
start(solve)
```

Можно обойтись и без логической операции *XOR*, если учесть, что два числа должны иметь **разную** чётность:

```
v = (a % 2 != 0) != (b % 2 != 0)
```

Ещё один способ решения задачи - импортировать **функцию-оператор *xor***:

```
from operator import xor
```

И использовать для решения задачи:

```
v = xor((a % 2 != 0), (b % 2 != 0))
```

Задание Boolean11

Ещё более коварная задача:

```
Даны два целых числа: A, B. Проверить истинность высказывания:
«Числа A и B имеют одинаковую четность».

Исходные данные
A = -5                                  B = -1

Пример верного решения
True
```

Простые высказывания одновременно должны быть либо ложными, либо истинными. Если же одно из выражений истинное, а второе ложное, то и сложное выражение должно быть ложным. Если мы применим к простым выражениям логическую операцию *ИСКЛЮЧАЮЩЕЕ ИЛИ*, то получим то, что нам нужно, но наоборот. А «наоборот» легко исправить с помощью оператора логического отрицания **not**.

Вот это настоящая логика!

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean11")

    #числа:
    a = get_int()
    b = get_int()

    #высказывание:
    #v = ((a % 2) + (b % 2)) % 2 == 0
    v = not ((a % 2 != 0) ^ (b % 2 != 0))
    put(v)


start(solve)
```

Но можно записать высказывание иначе:

```
v = (a % 2 != 0) == (b % 2 != 0)
```

Подумайте, правильно ли записано такое высказывание:

```
v = (a % 2) == (b % 2)
```

Задание Boolean12

Более длинная задача, чем мы решали раньше, но очень легкая:

Даны три целых числа: A, B, C. Проверить истинность высказывания:
«Каждое из чисел A, B, C положительное».

Исходные данные

A = -7 B = 9 C = 3

Пример верного решения

False

Все 3 простых высказывания **одновременно** должны быть верными, поэтому их нужно соединить в сложное выражение с помощью операции **ЛОГИЧЕСКОГО И**:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean12")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    #высказывание:
    v = (a > 0) and (b > 0) and (c > 0)
    put(v)


start(solve)
```

Задание Boolean13

Почти такое же задание, что и предыдущее, но на логическую операцию *ЛОГИЧЕСКОЕ ИЛИ*:

Даны три целых числа: A, B, C. Проверить истинность высказывания:
«Хотя бы одно из чисел A, B, C положительное».

Исходные данные

A = -3 B = -5 C = -5

Пример верного решения

False

В предыдущем коде нужно заменить ключевой слово **and** ключевым словом **or** – и это всё:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean13")
```

```
    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    #высказывание:
    v = (a > 0) or (b > 0) or (c > 0)
    put(v)


start(solve)
```

Задание Boolean14

Совсем интересная задача:

Даны три целых числа: A, B, C. Проверить истинность высказывания:
«Ровно одно из чисел A, B, C положительное».

Исходные данные

A = 7 B = 5 C = 8

Пример верного решения

False

В данном случае сложное выражение получается не только сложным, но и длинным, поэтому мы пойдём другим путём.

Объявим переменную **sum**:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean14")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    sum = 0
```

И будем добавлять к её значению 1, если число положительное:

```
if a > 0:
    sum += 1
if b > 0:
    sum += 1
if c > 0:
    sum += 1
```

Теперь по окончательному значению переменной *sum* мы легко определим, сколько было положительных чисел:

```
    #высказывание:
    v = sum == 1
    put(v)


start(solve)
```

Задание Boolean15

Как говорится, найдите разницу:

Даны три целых числа: A, B, C. Проверить истинность высказывания:
«Ровно два из чисел A, B, C являются положительными».

Исходные данные

A = -9 B = -3 C = -7

Пример верного решения

False

А разница такая: теперь нам нужны **2** положительных числа:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean15")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()
```

```
    sum = 0
    if a > 0:
        sum += 1
    if b > 0:
        sum += 1
    if c > 0:
        sum += 1

    #высказывание:
    v = sum == 2
    put(v)


start(solve)
```

Задание Boolean16

Несложное задание на сложное логическое выражение:

Дано целое положительное число. Проверить истинность высказывания:
«Данное число является четным двузначным».

Исходные данные

77

Пример верного решения

False

Для начала найдём простые высказывания:

- число чётное
- число двузначное

Так как простые высказывания **одновременно** должны быть истинными, то они должны соединяться в сложном выражении логической операцией *ЛОГИЧЕСКОЕ И*.

Если мы обозначим число буквой **a**, то первое высказывание можно записать так:

```
a % 2 == 0
```

Если число *a* чётное, то высказывание истинное.

Со вторым высказыванием сложнее. Мы легко определим, что число **двузначное**, если оно состоит из двух цифр. Но как это сделает компьютер? – Так как компьютер охотнее считает, чем думает, то мы можем предложить ему такой вариант определения двузначности заданного числа:

*Если число больше 9 **И** меньше 100, то оно двузначное.*

Эти условия должны выполняться **одновременно**, поэтому выражения *(a > 9)* и *(a < 100)* нужно соединить операцией **ЛОГИЧЕСКОГО И**.

Осталось собрать все наши рассуждения вместе – и задача решена:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean16")

    #число:
    a = get_int()

    #высказывание:
    v = (a % 2 == 0) and (a > 9) and (a < 100)
    put(v)


start(solve)
```

Логическое выражение можно упростить, если использовать специфические возможности *Питона*:

```
v = (a % 2 == 0) and (9 < a < 100)
```

Задание Boolean17

Задача, которая легко решается после предыдущей:

Дано целое положительное число. Проверить истинность высказывания:
«Данное число является нечетным трехзначным».

Исходные данные

9350

Пример верного решения

False

Нечётность числа мы определяем, как обычно, а **трёхзначность** так:

*Если число больше или равно 100 **И** меньше или равно 999, то оно трёхзначное.*

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean17")

    #число:
    a = get_int()

    #высказывание:
    v = (a % 2 != 0) and (a >= 100) and (a <= 999)
    put(v)


start(solve)
```

Можно использовать и **строгие** неравенства:

```
(a > 99) and (a < 1000)
```

но они не так хороши, потому что в них присутствуют нетрёхзначные числа.

Опять упрощаем выражение:

```
v = (a % 2 != 0) and (100 <= a <= 999)
```

Задание Boolean18

Для этой задачи можно придумать очень длинное логическое выражение:

Проверить истинность высказывания:
«Среди трех данных целых чисел есть
хотя бы одна пара совпадающих».

Исходные данные		
2	0	8
Пример верного решения		
False		

Поэтому мы опять пускаемся на хитрость и находим сумму совпадающих пар чисел. К счастью, из трёх чисел можно составить всего 3 пары, так что работы у нас немного:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean18")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    sum = 0
    if a == b:
        sum += 1
    if b == c:
        sum += 1
    if c == a:
        sum += 1

    #высказывание:
    v = sum > 0
    put(v)


start(solve)
```

Если хотя бы одна пара чисел совпадает, то значение переменной **sum** будет больше нуля.

Задание Boolean19

По сути, та же самая задача, что и предыдущая:

Проверить истинность высказывания:
«Среди трех данных целых чисел есть
хотя бы одна пара взаимно противоположных».

Исходные данные

-1	-1	-4

Пример верного решения

False

Взаимно противоположными называются числа, равные по абсолютной величине, но имеющие разные знаки:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean19")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    sum = 0
    if a == -b:
        sum += 1
    if b == -c:
        sum += 1
    if c == -a:
        sum += 1

    #высказывание:
    v = sum > 0
    put(v)


start(solve)
```

Так как сумма взаимно противоположных чисел равна нулю, то можно использовать и такое выражение:

```
if a + b == 0:
    sum += 1
```

Задание Boolean20

Более серьёзная задача:

Дано трехзначное число. Проверить истинность высказывания:
«Все цифры данного числа различны».

Исходные данные

777

Пример верного решения

False

Так как число трёхзначное, то оно состоит из трёх цифр, которые нужно предварительно из него извлечь.

Как это сделать, мы уже подробно разобрали в предыдущем наборе заданий:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean20")

    #число:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10

    #число сотен:
    s = sde // 100
```

Остальные действия очень простые.

Если все цифры различны, то ни одна пара цифр **не совпадает**:

```
    sum = 0
    if s == d:
        sum += 1
    if s == e:
        sum += 1
    if d == e:
        sum += 1

    #высказывание:
    v = sum == 0
    put(v)


start(solve)
```

Задание Boolean21

Более сложная задача с трёхзначными числами:

Дано трехзначное число. Проверить истинность высказывания:
«Цифры данного числа образуют возрастающую последовательность».

Исходные данные	Пример верного решения
138	True

Предварительно выделяем из заданного числа его цифры:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean21")

    #число:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10
```

```
    #число сотен:
    s = sde // 100
```

Если цифры образуют **возрастающую** последовательность, то каждая следующая больше предыдущей. То есть число десятков больше числа сотен, а число единиц больше числа десятков:

```
    #высказывание:
    v = (s < d) and (d < e)
    put(v)


start(solve)
```

Упрощённое выражение:

```
v = (s < d < e)
```

Задание Boolean22

Опять наши любимые трёхзначные числа:

Дано трехзначное число. Проверить истинность высказывания:
«Цифры данного числа образуют возрастающую или убывающую последовательность».

Исходные данные

270

Пример верного решения

False

Первую половину этой задачи мы уже решили.

Если же цифры образуют **убывающую** последовательность, то знак **<** в простых выражениях нужно заменить знаком **>**. Но это ещё не всё! Две половинки задачи нужно соединить **логической операцией ИЛИ**:

```
# -*- coding: cp1251 -*-
from pt4 import *
```

```
def solve():
    task("Boolean22")

    #число:
    sde = get_int()

    #число единиц:
    e = sde % 10

    #число десятков:
    d = sde // 10 % 10

    #число сотен:
    s = sde // 100

    #высказывание:
    v = (s < d) and (d < e)
    v = v or ((s > d) and (d > e))
    put(v)


start(solve)
```

Упрощённое выражение:

```
v = (s < d < e) or (e < d < s)
```

Задание Boolean23

А в этом задании и вовсе четырёхзначное число!

Дано четырехзначное число. Проверить истинность высказывания:
«Данное число читается одинаково слева направо и справа налево».

Исходные данные

9119

Пример верного решения

True

Но процесс выделения цифр из числа вы уже хорошо изучили, поэтому переходим к логическому выражению:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean23")

    #число:
    tsde = get_int()

    #число единиц:
    e = tsde % 10

    #число десятков:
    d = tsde // 100 % 10

    #число сотен:
    s = tsde // 10 % 10

    #число тысяч:
    t = tsde // 1000
```

Глядя на рисунок в начале задания, вы легко сообразите, что, если число одинаково читается в обе стороны, то число тысяч в нём равно числу единиц, а число сотен равно числу десятков:

```
    #высказывание:
    v = (t == e) and (s == d)
    put(v)


start(solve)
```

Задание Boolean24

Логико-математическая задача:

Даны числа A, B, C (число A не равно 0).
Рассмотрев **дискриминант** $D = B^2 - 4 \cdot A \cdot C$, проверить истинность высказывания:
«Квадратное уравнение $A \cdot x^2 + B \cdot x + C = 0$ имеет вещественные корни».

Исходные данные

A = -1.61 B = 3.72 C = -6.33

Пример верного решения

False

Чтобы её решить, необходимо вспомнить, когда квадратное уравнение имеет вещественные корни. Дискриминант вам в подмогу!

Итак, *если дискриминант неотрицательный, то квадратное уравнение имеет вещественные корни.*

Вычисляем дискриминант и составляем простейшее логическое выражение:

```
from pt4 import *
def solve():
    task("Boolean24")

    #числа:
    a = get_float()
    b = get_float()
    c = get_float()


    #дискриминант:
    d = b * b - 4 * a * c

    #высказывание:
    v = d >= 0
    put(v)


start(solve)
```

Задание Boolean25

Опять в логику вторгается математика:

Даны числа x, y. Проверить истинность высказывания:
«Точка с координатами (x, y) лежит во второй координатной четверти».

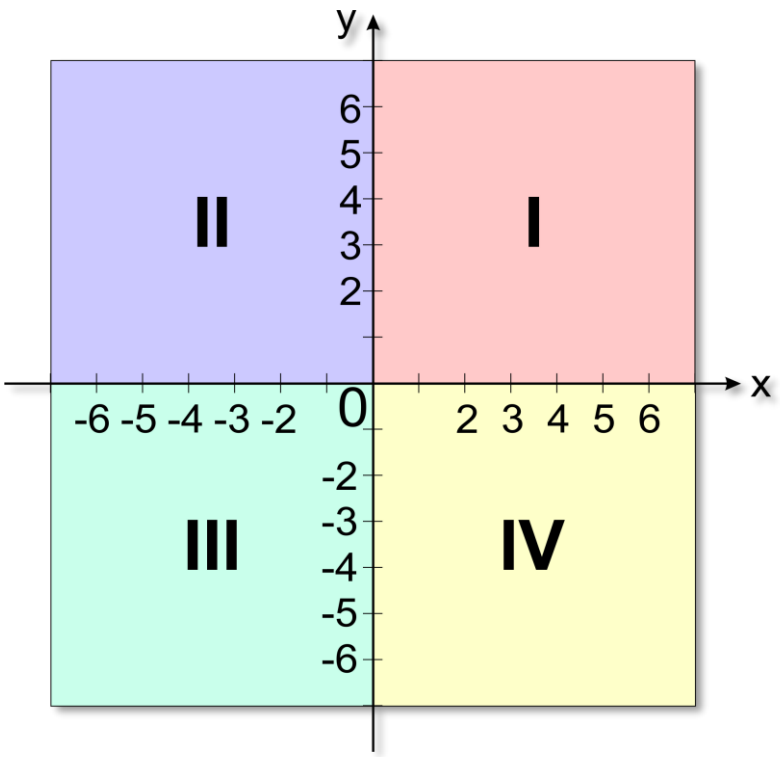
Исходные данные

x = -6.72 y = 8.52

Пример верного решения

True

Прежде всего, найдите на рисунке **вторую** координатную четверть!



Если точка лежит на ней (включая оси координат), то должны **одновременно** выполняться условия:

```
x <= 0
y >= 0
```

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean25")
```

```
    #числа:
    x = get_float()
    y = get_float()

    #высказывание:
    v = (x <= 0) and (y >= 0)
    put(v)


start(solve)
```

Логика должна быть железной!

Задание Boolean26

Это задание исключительно на внимательность:

Даны числа x, y. Проверить истинность высказывания:
«Точка с координатами (x, y) лежит в четвертой координатной четверти».

Исходные данные

x = 8.72 y = -8.24

Пример верного решения

True

Чтобы решить её, достаточно внимательно посмотреть на рисунок координатных четвертей из предыдущего задания:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean26")

    #числа:
    x = get_float()
    y = get_float()

    #высказывание:
    v = (x >= 0) and (y <= 0)
    put(v)


start(solve)
```

Задание Boolean27

Продолжаем четвертовать плоскость:

Даны числа x, y. Проверить истинность высказывания:
«Точка с координатами (x, y) лежит во второй
или третьей координатной четверти».

Исходные данные

x = 6.28 y = 4.09

Пример верного решения

False

Нужны ли комментарии к этой задаче? – Вопрос риторический...

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean27")

    #числа:
    x = get_float()
    y = get_float()

    #высказывание:
    v = x <= 0
    put(v)


start(solve)
```

Задание Boolean28

А вот тут нужна сообразительность:

Даны числа x, y. Проверить истинность высказывания:
«Точка с координатами (x, y) лежит в первой
или третьей координатной четверти».

Исходные данные

x = 8.04 y = 0.89

Пример верного решения

True

Заметим, что произведение координат любой точки в первой и третьей четвертях **неотрицательное** – и вся задача:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean28")

    #числа:
    x = get_float()
    y = get_float()

    #высказывание:
    v = x * y >= 0
    put(v)


start(solve)
```

Задание Boolean29

Задача из аналитической геометрии:

Даны числа x, y, x_1, y_1, x_2, y_2. Проверить истинность высказывания:
«Точка с координатами (x, y) лежит внутри прямоугольника, левая верхняя вершина которого имеет координаты (x_1, y_1), правая нижняя – (x_2, y_2), а стороны параллельны координатным осям».

Исходные данные

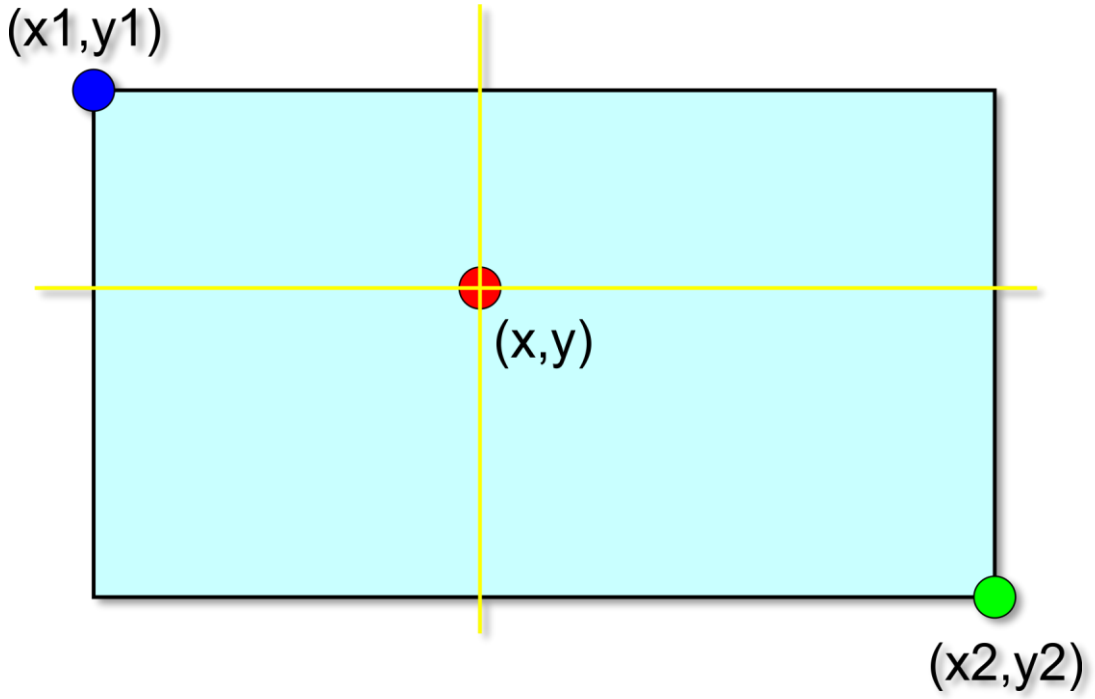
x, y: -2.00 -6.00

x_1, y_1: -3.00 -3.00 x_2, y_2: 1.00 -9.00

Пример верного решения

True

Чтобы решить задачу, нужно сначала вычертить прямоугольник и поставить в произвольном месте внутри него точку с координатами *(x, y)*:



На рисунке хорошо видно, что для такой точки **одновременно** должны выполняться условия:

x > x1
x < x2
y < y1
y > y2

Составить из этих простых логических выражений одно сложное как раз несложно:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean29")

    #числа:
    x = get_float()
    y = get_float()
    x1 = get_float()
    y1 = get_float()
    x2 = get_float()
    y2 = get_float()

    #высказывание:
    v = (x > x1) and (x < x2) and (y < y1) and (y > y2)
    put(v)


start(solve)
```

Упрощённое выражение:

```
v = (x1 < x < x2) and (y2 < y < y1)
```

Задание Boolean30

Очередная задача из геометрии:

Даны целые числа a, b, c, являющиеся сторонами некоторого треугольника.
Проверить истинность высказывания:
«Треугольник со сторонами a, b, c является равносторонним».

Исходные данные

a = 13 b = 26 c = 16

Пример верного решения

False

Так как длины всех сторон треугольника нам известны, то нужно только **попарно** сравнить их между собой. Из трёх сторон можно составить 3 различные пары, но достаточно сравнить 2 пары сторон, так как третья пара в случае успеха совпадёт автоматически:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean30")

    #числа:
    a = get_int()
    b = get_int()
    c = get_int()

    #высказывание:
    v = (a == b) and (b == c)
    put(v)


start(solve)
```

Задание Boolean31

Равносторонний треугольник из предыдущего задания - это частный случай равнобедренного треугольника, поэтому на этот раз мы должны решить более общую задачу:

Даны целые числа a, b, c, являющиеся сторонами некоторого треугольника.
Проверить истинность высказывания:
«Треугольник со сторонами a, b, c является равнобедренным».

Исходные данные

a = 15 b = 1 c = 15

Пример верного решения

True

Итак, треугольник является равнобедренным, если совпадает длина **любой** пары его сторон:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean31")

    #длины сторон:
```

```
    a = get_int()
    b = get_int()
    c = get_int()

    #высказывание:
    v = (a == b) or (a == c) or (b == c)
    put(v)


start(solve)
```

То есть иногда среди равнобедренных треугольников будут попадаться и равносторонние.

Задание Boolean32

Переходим к прямоугольным треугольникам, которые могут быть равнобедренными, но не равносторонними.

Даны целые числа a, b, c, являющиеся сторонами некоторого треугольника.
Проверить истинность высказывания:
«Треугольник со сторонами a, b, c является прямоугольным».

Исходные данные

a = 16 b = 8 c = 20

Пример верного решения

False

При решении этой задачи нам не обойтись без теоремы Пифагора про его знаменитые штаны.

Из неё следует, что *в прямоугольном треугольнике квадрат гипотенузы равен сумме квадратов катетов*.

Так как мы не знаем, какая из сторон является в треугольнике гипотенузой, а отыскивать её – дополнительная задача, то мы просто последовательно «провозгласим» каждую из сторон гипотенузой, тогда двум оставшимся достанется роль катетов:

```
# -*- coding: cp1251 -*-
from pt4 import *
```

```
def solve():
    task("Boolean32")

    #длины сторон:
    a = get_int()
    b = get_int()
    c = get_int()

    #квадраты длин:
    a2 = a * a
    b2 = b * b
    c2 = c * c

    #высказывание:
    v = (a2 == b2 + c2) or (b2 == a2 + c2) or (c2 == a2 + b2)
    put(v)


start(solve)
```

Задание Boolean33

Очередная треугольная задача:

Даны целые числа a, b, c.
Проверить истинность высказывания:
«Существует треугольник со сторонами a, b, c».

Исходные данные

a = 24 b = 17 c = 26

Пример верного решения

True

Во-первых, длины всех сторон должны быть положительны. Так как далеко не все целые числа положительны, то необходимо проверить числа **a**, **b**, **c**:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean33")

    #длины сторон:
    a = get_int()
    b = get_int()
```

```
    c = get_int()

    #высказывание:
    v = (a > 0) and (b > 0) and (c > 0)
```

Во-вторых, не всякие три числа могут выражать длину сторон треугольника. Это возможно только в том случае, если сумма любых двух чисел больше третьего. Из этого условия следует, что должны **одновременно** выполняться логические выражения:

$$a < b + c$$
$$b < a + c$$
$$c < a + b$$

Кроме того, если вы помните, непременно должно быть истинным и первое высказывание. Поэтому сложное высказывание должно быть таким:

```
    v = v and ((a < b + c) and (b < a + c) and (c < a + b))
    put(v)


start(solve)
```

Задание Boolean34

Шахматная доска раскрашена в 2 цвета:

Даны координаты поля шахматной доски x, y (целые числа, лежащие в диапазоне 1-8). Учитывая, что левое нижнее поле доски (1,1) является черным, проверить истинность высказывания: «Данное поле является белым».

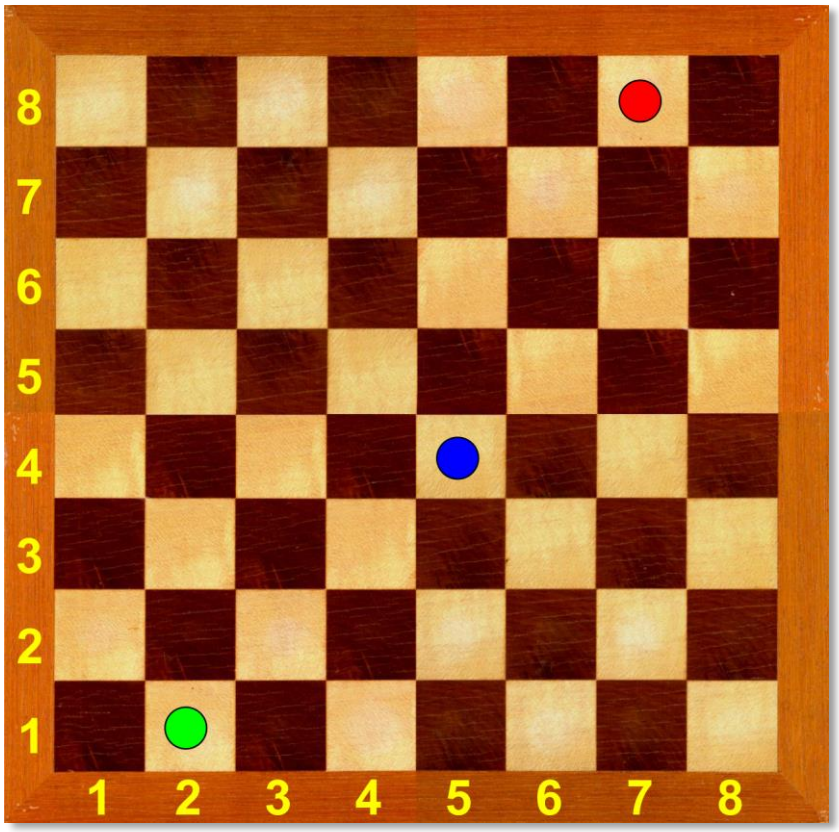
Исходные данные

x, y: 6 3

Пример верного решения

True

Если **внимательно** и **вдумчиво** посмотреть на доску, то можно заметить, что *сумма координат белых клеток всегда нечётная*:




```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean34")

    #координаты клетки:
    x = get_int()
    y = get_int()

    #высказывание:
    v = (x + y) % 2 != 0
    put(v)


start(solve)
```

Задание Boolean35

Продолжаем раскрашивать шахматную доску:

Даны координаты двух различных полей шахматной доски x_1, y_1, x_2, y_2 (целые числа, лежащие в диапазоне 1-8). Проверить истинность высказывания: «Данные поля имеют одинаковый цвет».

Исходные данные

x_1, y_1: 4 8 x_2, y_2: 4 5

Пример верного решения

False

Из предыдущего задания следует, что клетки одинакового цвета имеют и одинаковую чётность суммы координат, то есть либо обе суммы чётные, либо обе суммы нечётные:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean35")

    #координаты клеток:
    x1 = get_int()
    y1 = get_int()
    x2 = get_int()
    y2 = get_int()

    #высказывание:
    v = (x1 + y1) % 2 == (x2 + y2) % 2
    put(v)


start(solve)
```

Задание Boolean36

Шахматная задача с ладейным уклоном:

Даны координаты двух различных полей шахматной доски x_1, y_1, x_2, y_2 (целые числа, лежащие в диапазоне 1-8). Проверить истинность высказывания: «Ладья за один ход может перейти с одного поля на другое».

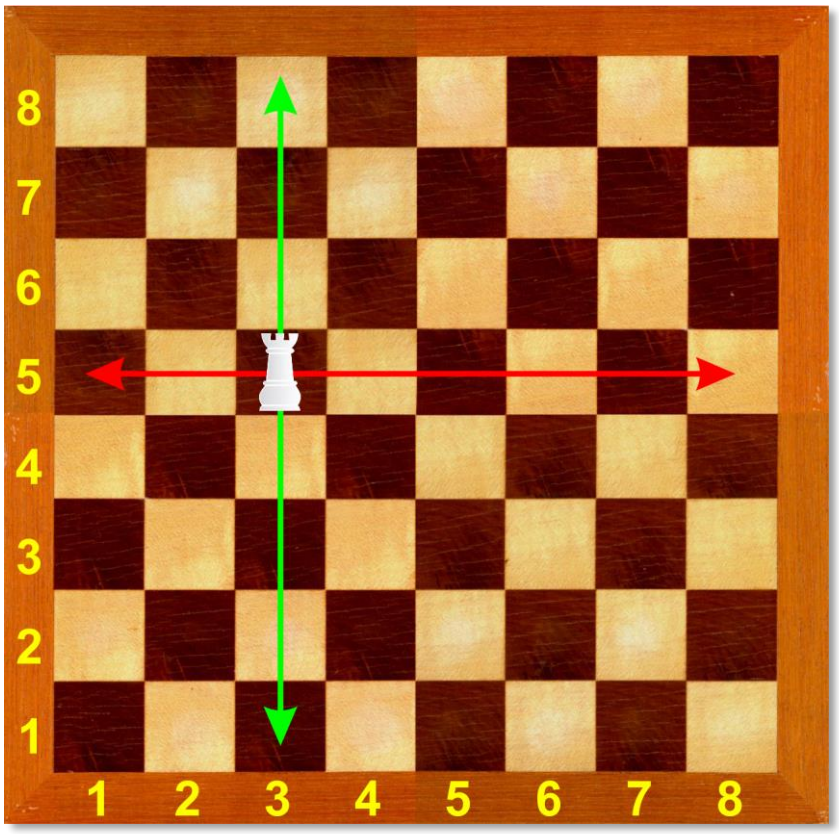
Исходные данные

x_1, y_1: 4 2 x_2, y_2: 4 3

Пример верного решения

True

Ладья – фигура тяжёлая и прямолинейная, то есть за 1 ход может переместиться только либо по горизонтали, либо по вертикали. Из этого вытекает, что заданные клетки должны иметь или одинаковые горизонтальные, или одинаковые вертикальные координаты:




```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean36")
```

```
    #координаты клеток:
    x1 = get_int()
    y1 = get_int()
    x2 = get_int()
    y2 = get_int()

    #высказывание:
    v = (x1 == x2) or (y1 == y2)
    put(v)


start(solve)
```

Задание Boolean37

Королевская задача:

Даны координаты двух различных полей шахматной доски x_1, y_1, x_2, y_2 (целые числа, лежащие в диапазоне 1-8). Проверить истинность высказывания: «Король за один ход может перейти с одного поля на другое».

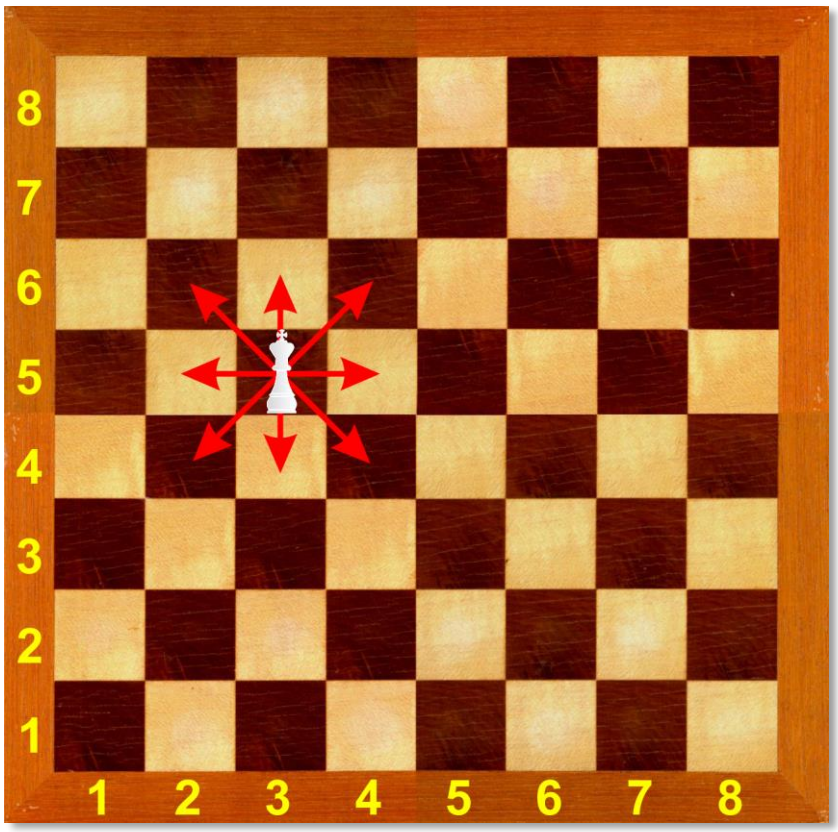
Исходные данные

x_1, y_1: 5 3 x_2, y_2: 1 6

Пример верного решения

False

Шахматный король ходит только на **соседнюю** клетку:



Координаты соседних клеток отличаются на 0 или на 1. Если король ходит прямо, то изменяется только 1 координата, а если криво – то обе.

Разность координат соседних клеток может быть как положительной, так и отрицательной, поэтому лучше использовать **абсолютную** величину разности:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean37")

    #координаты клеток:
    x1 = get_int()
    y1 = get_int()
    x2 = get_int()
    y2 = get_int()

    #высказывание:
    v = (abs(x1 - x2) <= 1) and (abs(y1 - y2) <= 1)
    put(v)



start(solve)
```

Задание Boolean38

В ход тяжёлой поступью пошли слоны:

Даны координаты двух различных полей шахматной доски x_1, y_1, x_2, y_2 (целые числа, лежащие в диапазоне 1-8). Проверить истинность высказывания: «Слон за один ход может перейти с одного поля на другое».

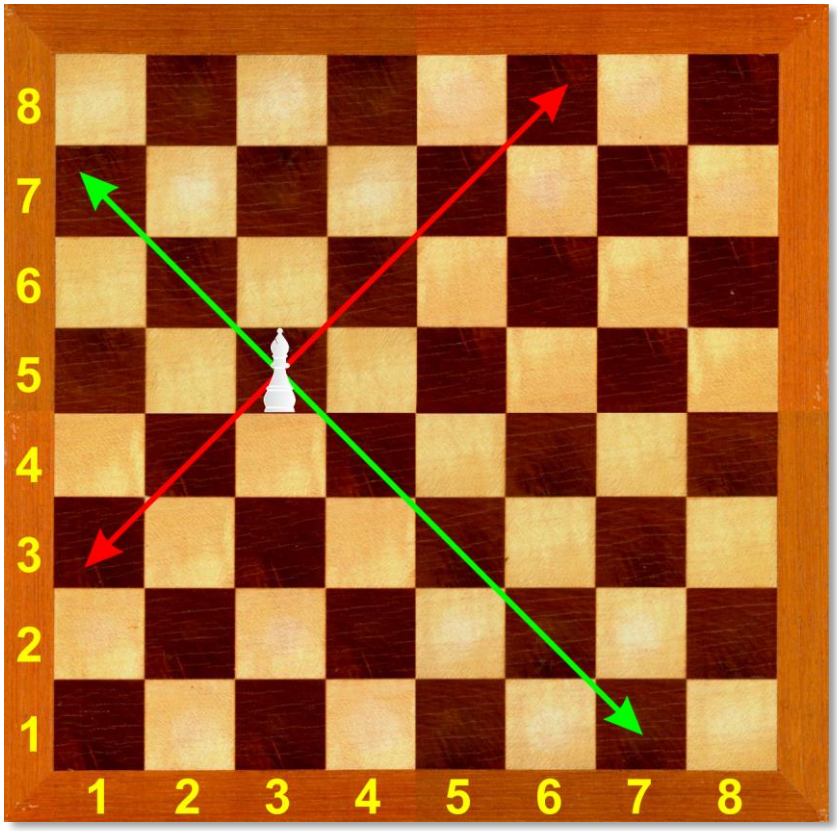
Исходные данные

x_1, y_1: 5 1 x_2, y_2: 1 5

Пример верного решения

True

Слоны могут ходить только по диагонали, но на любое число клеток, если они при этом не вываливаются с доски:



Легко заметить, что при перемещении по диагонали абсолютная величина разности координат по обеим осям должна быть одинаковой:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean38")
```

```
    #координаты клеток:
    x1 = get_int()
    y1 = get_int()
    x2 = get_int()
    y2 = get_int()

    #высказывание:
    v = abs(x1 - x2) == abs(y1 - y2)
    put(v)


start(solve)
```

Задание Boolean39

Ферзь, как известно, самая мощная шахматная фигура:

Даны координаты двух различных полей шахматной доски x_1, y_1, x_2, y_2 (целые числа, лежащие в диапазоне 1-8). Проверить истинность высказывания: «Ферзь за один ход может перейти с одного поля на другое».

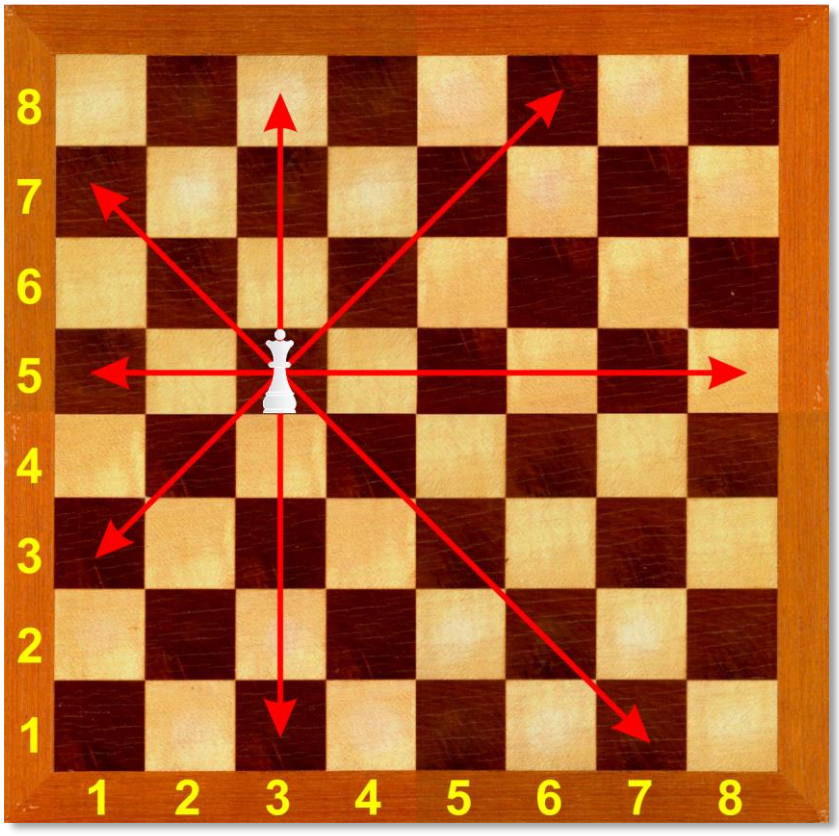
Исходные данные

x_1, y_1: 2 5 x_2, y_2: 2 6

Пример верного решения

True

Он может ходить, как ладья, - по горизонтали и по вертикали – или, как слон, – по диагонали:



Из этого следует, что нужно соединить условия для хода ладьи и хода слона логической операцией **ИЛИ**:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
```

```
    task("Boolean39")

    #координаты клеток:
    x1 = get_int()
    y1 = get_int()
    x2 = get_int()
    y2 = get_int()

    #высказывание:
    v = (abs(x1 - x2) == abs(y1 - y2)) or (x1 == x2) or (y1 == y2)
    put(v)


start(solve)
```

Задание Boolean40

Конь – самая коварная шахматная фигура:

Даны координаты двух различных полей шахматной доски x_1, y_1, x_2, y_2 (целые числа, лежащие в диапазоне 1-8). Проверить истинность высказывания: «Конь за один ход может перейти с одного поля на другое».

Исходные данные

x_1, y_1: 4 1 x_2, y_2: 3 3

Пример верного решения

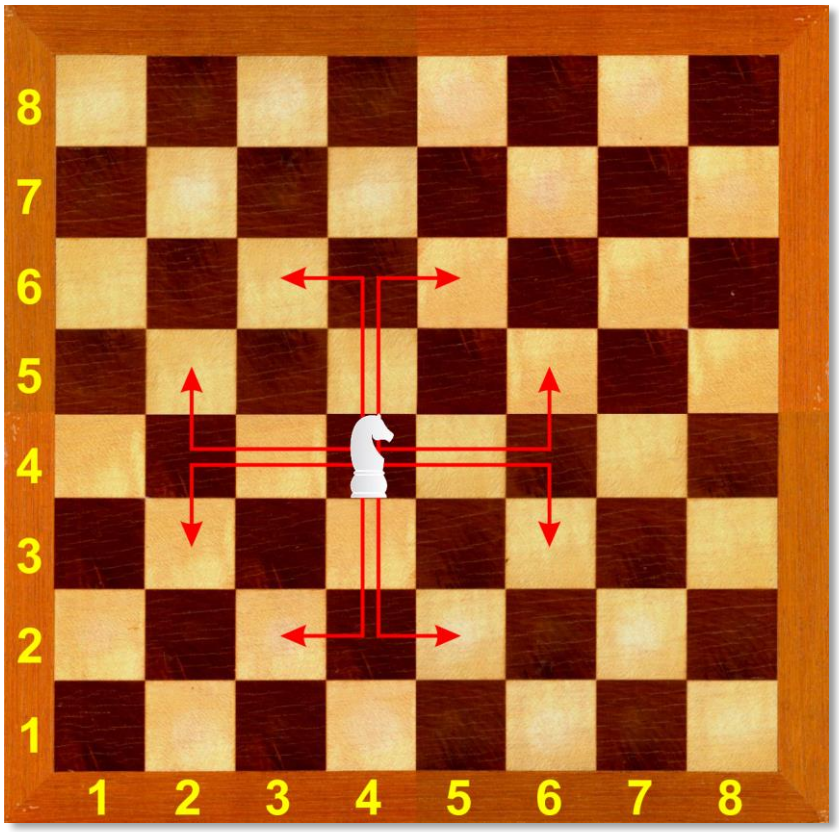
True

Он может неожиданно появиться из-за угла и больно ударить копытом. Недаром бытует выражение *сделать ход конём*, которое означает хитрый, неожиданный ход.

Из песни Высоцкого *Честь шахматной короны* мы знаем, что кони ходят только так:

Помню - всех главнее королева:
Ходит взад-вперёд и вправо-влево, -
Ну а кони вроде - буквой «Г».

У этой буквы одна сторона – 3 клетки, а вторая – 2. Так ходить, конечно, и коню трудно, и нам вычислять координаты клеток нелегко:



Так как шахматы предполагают размышления над каждым ходом, то давайте поразмышляем.

Координаты начальной и конечной клеток коня различаются на 2 и на 1. Это значит, что абсолютные величины разности координат всегда равны 2 и 1. Причём в зависимости от направления хода, б***о***льшая разность может быть у горизонтальных координат или у вертикальных. Однако произведение чисел 2 и 1 **всегда** равно **двум**. Поэтому конская задача имеет очень простое решение:

```
# -*- coding: cp1251 -*-
from pt4 import *
def solve():
    task("Boolean40")

    #координаты клеток:
    x1 = get_int()
    y1 = get_int()
    x2 = get_int()
    y2 = get_int()

    #высказывание:
    v = abs(x1 - x2) * abs(y1 - y2) == 2
    put(v)
```

```
start(solve)
```

Если вы, конечно, умеете делать ход конём!

Литература

Серия Учись программировать с *Питоном*

[Python01]

Рубанцев Валерий

Решебник на языке Python к Электронному задачнику Абрамяна
180 с.

В эту книгу включены задания трёх первых наборов: **Begin**, **Integer**, **Boolean**. Всего 110 заданий. Разработка программ ведётся в удобной среде *PyCharm*, но можно решать задачи и в более простой среде *IDLE*.

Этот решебник можно считать первым изданием книги *Решаем задачи Абрамяна на языке Питон: Begin, Integer, Boolean, If, Case*. В ней заданий меньше, но они проверяются автоматически, что очень важно для начинающих программистов. Решения всех задач подробно и понятно объясняются. Незаменимая книга для самостоятельного изучения языка *Питон* и для подготовки к *ЕГЭ*.

[Python02]

Рубанцев Валерий

Развивающее программирование
Практикум по решению задач на языке Питон 3. Базовый уровень
500 с.

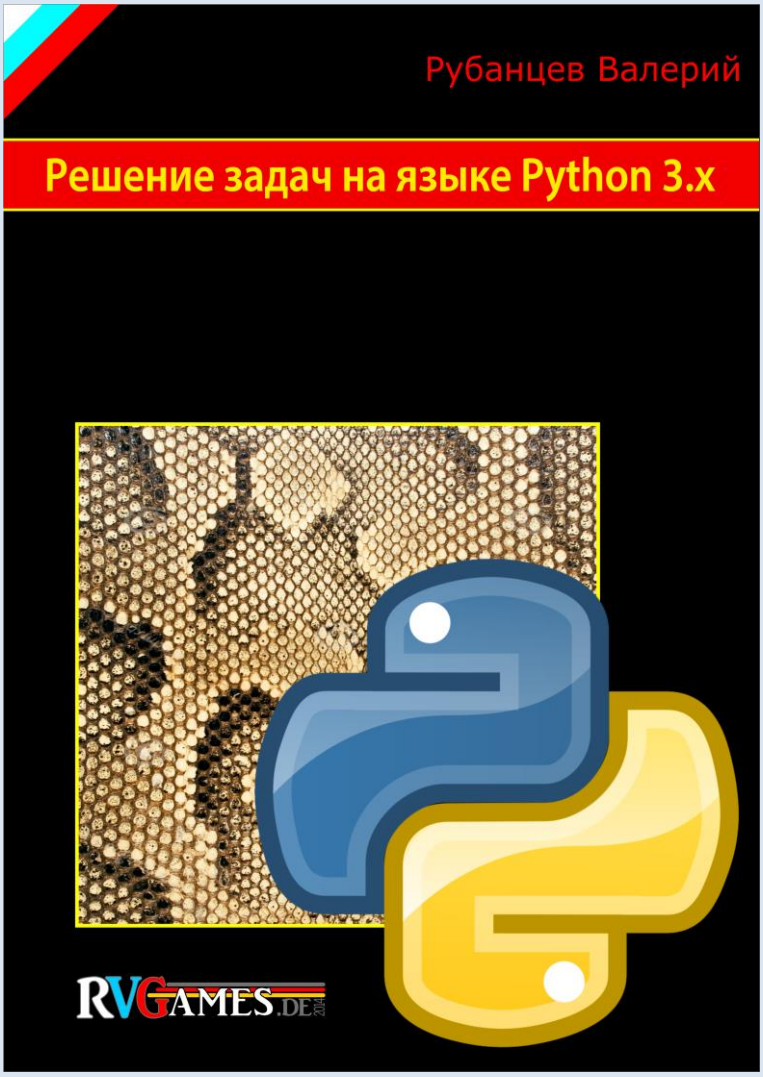
В книге подробно рассматривается решение более 100 задач: математических, словесных, комбинаторных, вероятностных, игровых.

Везде используется версия *Python 3.5*, но исходный код будет работать и в
более ранних версиях *Питона* (но не в 2.7!).

Лучшие упражнения для отработки навыков программирования на языке *Питон*.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *Питоне*.

[Python03]



Рубанцев Валерий

Решение задач на языке *Python 3.x*
390 с.

В этой книге подробно рассматривается решение **90 задач**: математических, комбинаторных, вероятностных, игровых.

Несмотря на сравнительно небольшой объём книги, она охватывает все ключевые элементы языка *Питон*. В самом её начале вы найдёте *Тематический указатель*, который поможет вам ориентироваться во всех проектах и легко находить нужный. В конце многих глав имеются задания для *самостоятельного решения*.

[Python04]

Рубанцев Валерий

Решаем задачи Абрамяна на языке Питон: Begin, Integer, Boolean, If, Case
Серия: *Учись программировать с Питоном*
230 с.

Решение задач первых пяти наборов из *Задачника Абрамяна.*

Решения всех задач подробно и понятно объясняются. Незаменимая книга для самостоятельного изучения языка *Питон* и для подготовки к *ЕГЭ*.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *Питоне*.

[Python05]

Рубанцев Валерий

Решаем задачи Абрамяна на языке Питон: For, While, Series, Func, Matrix
Серия: *Учись программировать с Питоном*
270 с.

Решение пяти наборов задач из *Задачника Абрамяна.*

Решения всех задач подробно и понятно объясняются. Незаменимая книга для самостоятельного изучения языка *Питон* и для подготовки к *ЕГЭ*.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *Питоне*.

Серия Учись программировать с *Котлином*

[Kotlin01]

Рубанцев Валерий

Привет, Котлин!
Прогграммирование на языке Котлин для детей

RVGames, 2018. – 350 с.

Это **самоучитель по программированию** на языке *Котлин* для детей, начиная с 10-летнего возраста. Для книги отобрано ровно столько учебного материала, сколько его необходимо и достаточно, чтобы начать писать программы на языке *Котлин* самостоятельно.

[Kotlin02]

Рубанцев Валерий

Практикум по решение задач на языке Kotlin для детей

RVGames, 2018. – 200 с.

В этой книге собраны занимательные математические задачи, которые можно решать на языке *Котлин*. Решение всех задач подробно описано. В них используются все теоретические знания, которые читатели получили в первой книге. Решение задач – отличная тренировка при изучении нового языка программирования!

[Kotlin03]

Рубанцев Валерий

Практикум по решению задач на языке Kotlin для школьников
RVGames, 2018. – 250 с.

Решаем занимательные математические задачи всех времён и народов на языке *Котлин* в среде *IntelliJ IDEA*. При решении задач используются все базовые понятия языка *Котлин*.

Совершенно необходимая книга для закрепления знаний и укрепления навыков программирования на языке *Котлин*.

[Kotlin05]

Рубанцев Валерий

Решение задач на языке Kotlin
RVGames, 2018. – 370 с.

Решение математических задач на языке *Котлин* в среде *IntelliJ IDEA*. При решении задач используются все базовые понятия языка *Котлин*.

Совершенно необходимая книга для закрепления знаний и укрепления навыков программирования на языке *Котлин*.

[Kotlin06]

Рубанцев Валерий

Основы компьютерной графики на языке *Котлин*

RVGames, 2018. – 500 с.

Это **самоучитель по компьютерной графике** для начинающих. В книге подробно описаны возможности графической библиотеки *core.js* – основы языка *Процессинг* - для простой и эффективной разработки графических приложений на языке *Котлин*.

Эта книга поможет вам быстро изучить основы компьютерной графики, и вы сможете самостоятельно рисовать красивые узоры, решать задачи, писать игры и разрабатывать компьютерные модели по биологии, физике, химии.

В ней вы найдёте исчерпывающий теоретический материал для самостоятельного и разностороннего творчества.

Серия Учись программировать с *Процессингом*

[Processing 03]

Рубанцев Валерий

Учись программировать с Процессингом

RVGames, 2018. – 540 с.

Это **самоучитель по программированию** на языке *Ява* для начинающих.
Для книги отобрано ровно столько учебного материала, сколько его необходимо и достаточно, чтобы писать программы школьного уровня.

[Processing 04]

Рубанцев Валерий

Учись программировать с Процессингом
Основы компьютерной графики

RVGames, 2018. – 490 с.

Это **самоучитель по компьютерной графике** для начинающих.

Эта книга поможет вам быстро изучить основы компьютерной графики, и вы сможете самостоятельно рисовать красивые узоры, решать задачи, писать игры и разрабатывать компьютерные модели по биологии, физике, химии.

В ней вы найдёте исчерпывающий теоретический материал для самостоятельного и разностороннего творчества.

Серия Программирование на *ЯваСкрипте*

[JavaScript 01]

Рубанцев Валерий

Программирование на *ЯваСкрипте*
Занимательная графика на *ЯваСкрипте*

RVGames, 2017. – 430 с.

В книге подробно описываются возможности графической библиотеки *p5.js* для простой и эффективной разработки графических приложений на языке *ЯваСкрипт.*

Все функции проиллюстрированы многочисленными примерами.

Рубанцев Валерий

Тотальный тренинг по *ЯваСкрипту*
Массивы и функциональное программирование

RVGames, 2017. – 260 с.

Книга о массивах, методах и функциональном программировании на *ЯваСкрипте*.

Все методы проиллюстрированы демонстрационными проектами.

На занимательных примерах показано, как решать практические задачи на *ЯваСкрипте* в функциональном стиле.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *ЯваСкрипте*.

[JavaScript 03]

Рубанцев Валерий

Тотальный тренинг по *ЯваСкрипту*
Простые компьютерные игры

RVGames, 2017. – 220 с.

В книге подробно описывается разработка 10 простых компьютерных игр. Среди них есть и очень известные игры – *Игра Баше*, *Угадай число*, *Закраска* – и не очень, и совсем новые – *Пузыри*, *Блиц-Клик*, *Охота на Скалоеда* и *Скалоедов*, две программы про Незнайку и великолепная головоломка *Ножки вверх*!

Цель книги: научиться писать простые компьютерные игры на языке ЯваСкрипт с использованием графической библиотеки p5.js.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *ЯваСкрипте*.

[JavaScript 04]

Рубанцев Валерий

Тотальный тренинг по *ЯваСкрипту*
Простые компьютерные игры

RVGames, 2017. – 310 с.

Продолжаем программировать компьютерные игры и головоломки: *Солитер*, *Прыгающие лягушки*, *Крестики-нолики*, математическая *Игра Ярбро*, головоломки **Eliminator** и *Местор*

Научимся использовать в программах анимацию и метод минимакса, разрабатывать графический интерфейс, писать "решалки" для головоломок, придумывать свои уровни и программы.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на ЯваСкрипте.

[JavaScript 05]

Рубанцев Валерий

Программирование на *ЯваСкрипте*
Программирование игр для детей

RVGames, 2017. – 240 с.

Учимся программировать интересные и красочные компьютерные игры для детей! Самая известная из них – *Фрудоку*. Это упрощённый вариант судоку 6 х 6 клеточек с фруктами вместо цифр. Менее известны, но не менее увлекательны игры *ABCD*, *Arukone*, *Bit-Shift*, *Grand Tour*, *Кубиковая считалка* и *Римская задача*.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *ЯваСкрипте*.

[JavaScript 06]

Рубанцев Валерий

Программирование на *ЯваСкрипте*
Программирование детских игр

RVGames, 2017. – 230 с.

Новые компьютерные игры и головоломки для детей! **Фокус**, **Собиратель монет**, **Собиратель букв**, **Анаграммы**, **Коровы**, **Сикаку**, **Тетрамино**, **Тетраминки**.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на ЯваСкрипте.

[JavaScript 07]

Рубанцев Валерий

Программирование на *ЯваСкрипте*
Классические компьютерные игры

RVGames, 2017. – 250 с.

Совершенствуем и развиваем навыки программирования компьютерных игр на *ЯваСкрипте*.

На этот раз на очень занимательных примерах: **Тетрис**, **Змейка**, **Сапёр**, **Bubble Shooter**! А также: мы разовьём плодотворные классические идеи и напишем клоны: игры **Рекстрис**, **По грибы** и **вторую Змейку**.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *ЯваСкрипте*.

[JavaScript 08]

Рубанцев Валерий

Программирование на *ЯваСкрипте*
Компьютерные игры для начальной школы

RVGames, 2017. – 220 с.

Новые компьютерные игры и головоломки для детей! **Перекинь мостик**, **Сотенный билет**, **Найди треугольник**, **Психологическая считалка**, **Фруктосчёт**, **Раскрась карту**.

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *ЯваСкрипте*.

[JavaScript 09]

Рубанцев Валерий

Программирование на *ЯваСкрипте*
Новые компьютерные игры

RVGames, 2018. – 290 с.

В этой книжке только одна головоломная программа, а все остальные – это весёлые, заводные игры:

- Головоломка *Инь-Ян*
- Игра *Катапульта*
- Игра *Сквош*
- Игра *Сквош на двоих*
- Игра *Теннис*
- Игра *Танчики*
- Игра *Платформер*

Для учащихся, учителей информатики, любителей программирования и начинающих программистов, имеющих небольшой опыт в программировании на *ЯваСкрипте*.

Серия Программирование для детей

[Скретч01]

Рубанцев Валерий

Программирование для детей
Занимательные уроки со *Скретчем*

RVGames, 2016. – 260 с.

Занимательные проекты по различным школьным предметам и компьютерные игры.

[Скретч02]

Рубанцев Валерий

Программирование для детей
Новые занимательные уроки со *Скретчем*

RVGames, 2016. – 180 с.

Занимательные проекты по различным школьным предметам и компьютерные игры.

[Скретч03]

Рубанцев Валерий

Программирование для детей
Самые новые занимательные уроки со *Скретчем*

RVGames, 2016. – 180 с.

Занимательные проекты по различным школьным предметам и компьютерные игры.

[Скретч04]

Рубанцев Валерий

Программирование для детей
Лучшие занимательные уроки со *Скретчем*

RVGames, 2016. – 160 с.

Занимательные проекты по различным школьным предметам и компьютерные игры.

[Скретч05]

Рубанцев Валерий

Программирование для детей
Занимательные задачи со *Скретчем*

RVGames, 2016. – 160 с.

Решение занимательных математических задач на *Скретче*.

[Скретч06]

Рубанцев Валерий

Программирование для детей
Скретч в первом классе

RVGames, 2016. – 180 с.

В книге есть всё, чтобы успешно начать программировать на языке программирования *Скретч*.

Интересные проекты, ещё более интересные задачи и головоломки.

[Скретч07]

Рубанцев Валерий

Программирование для детей
Скретч во втором классе

RVGames, 2016. – 220 с.

Продолжаем изучать *Скретч* во втором классе! Программирование, арифметика, алгебра, геометрия.

Много проектов - учебных и игровых, компьютерная игра Раскраска, головоломки и интересные сведения о числах

[Геогебра]

Рубанцев Валерий

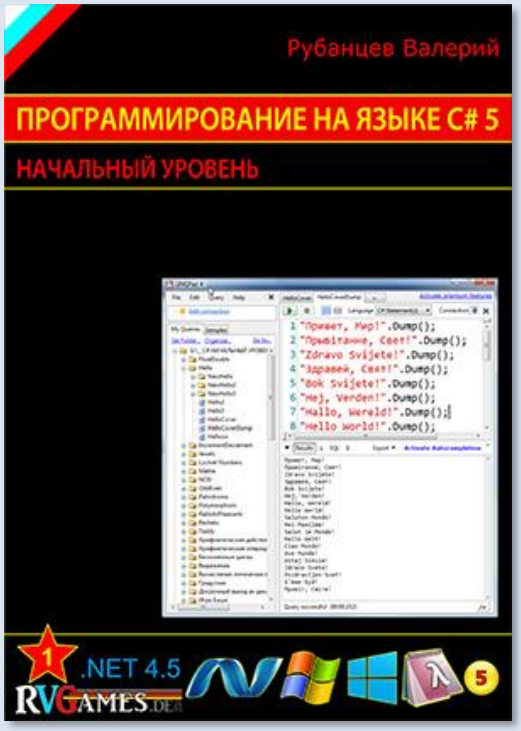
Высокие технологии в школе
Занимательные задачи с *Геогеброй*

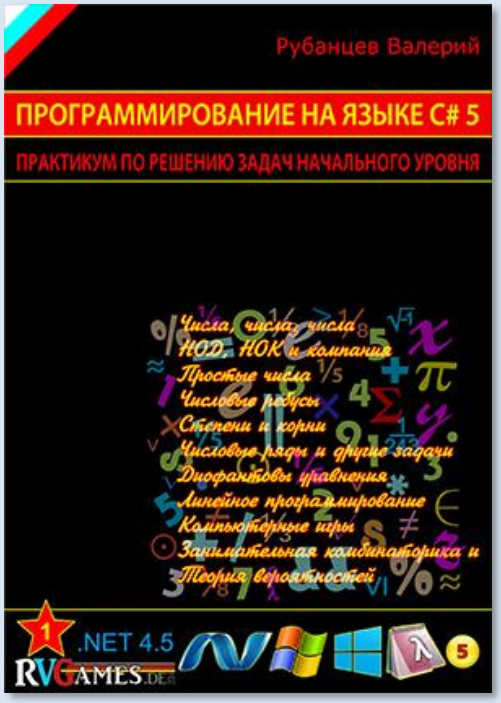
RVGames, 2016. – 160 с.

Решение занимательных математических задач в среде *Геогебра*.

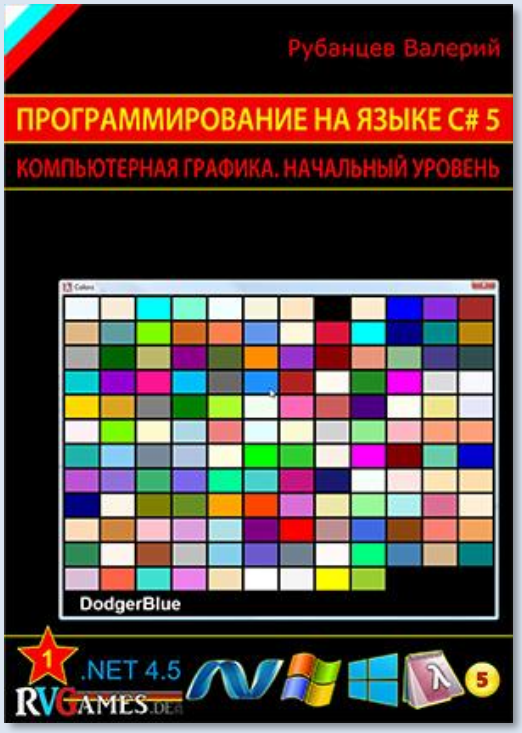
Несколько десятков интересных задачек по всем школьным разделам алгебры.

Серия Программирование на языке C# 5.0: Начальный уровень

[CS10]	
	*Рубанцев Валерий* **Программирование на языке C# 5:** Начальный уровень RVGames, 2014. – 620 с. Основы программирования на языке *Си-шарп*.

[CS11]	
	*Рубанцев Валерий* **Программирование на языке C# 5:** Практикум по решению задач начального уровня RVGames, 2014. – 420 с. Многочисленные проекты для укрепления навыков программирования на языке *Си-шарп*.

[CS12]



Рубанцев Валерий

Программирование на языке C# 5:
Компьютерная графика. Начальный уровень

RVGames, 2014. – 460 с.

Основы компьютерной графики.

[CS13]

Рубанцев Валерий

Программирование на языке C# 5:
Тотальный тренинг по *Си-шарпу*. Начальный уровень

RVGames, 2014. – 400 с.

Разнообразные полезные и занимательные проекты на языке *Си-шарп*.

[Питон]

Рубанцев Валерий

Развивающее программирование
Практикум по решению задач на языке *Питон 3*. Базовый уровень

RVGames, 2016. – 500 с.

В книге подробно рассматривается решение более 100 задач: математических, словесных, комбинаторных, вероятностных, игровых.

Лучшие упражнения для отработки навыков программирования на языке *Питон*.

[Си-шарп]

Рубанцев Валерий

Компьютер, наука и жизнь
Занимательные математические задачи: От древности до современности

RVGames, 2016. – 500 с.

Около 150 проектов на языке *Си-шарп,* показывающих, как можно решать разнообразные занимательные математические задачи на компьютере.