

Разбор задач конкурса "Звезды Кассиопеи"

Надобных Дмитрий, Пугач Сергей, Чечеватов Роман

20.05.2021

1 Ручные задачи

Для решения данных задач не нужны никакие навыки программирования и они вам не сильно помогут. Вместо этого участникам предлагалось научиться искать информацию в файлах, анализировать их.

1.1 МР5

Эта задача засчитывалась как решенная после перехода по ссылке. Ничего сложного.

1.2 Автостоп

Как сказано в условии, ответом на данную задачу является ответ на Главный вопрос жизни, вселенной и всего такого. Вопрос этот был задан английским писателем Дугласом Адамсом в книге "The Hitchhiker's Guide to the Galaxy", русским читателям известной как "Автостопом по галактике".

Участники, читавшие этот роман могут спокойно ответить 42 и получить баллы, остальным придется скопировать вопрос в Гугл и получить ответ в первом же результате.

Ответ: 42

1.3 Язык программирования

Для решения этой задачи участникам необходимо было изучить HTML-код страницы задачи. Для этого можно воспользоваться встроенными в браузер инструментами разработчика, или скачать страницу и открыть в текстовом редакторе.

Независимо от выбранного способа, ответ находился в комментарии рядом с формой ввода ответа.

```
<!DOCTYPE html>
<html lang="ru">
  <head>...</head>
  <body>
    <header>...</header>
    <!-- Очки за задание -->
    <div class="big-card">...</div>
    <!-- Описание задания -->
    <div class="big-card">...</div>
    <!-- Поля ввода ответа -->
    <div class="big-card">
      <h2>Ваш ответ</h2>
      <form method="post">
        <input name="answer" type="text">
        ... <!-- Просто не так ли? flag{html_is_the_best_language}-->
        <input type="submit" value="Отправить">
      </form>
    </div>
  </body>
</html>
```

Рисунок 1: HTML-код страницы задачи

Ответ: flag{html_is_the_best_language}

1.4 Фотосессия

В данной задаче участникам давался для анализа JPG-файл с картинкой. Однако, такие файлы зачастую содержат в себе не только данные изображения, но и дополнительную информацию. Например, дату съемки, модель камеры, имя автора. И среди этой дополнительной информации и был спрятан флаг.

Ответ: `flag{haha_you_thought_it_will_be_ccv}`

1.5 PI

Здесь участников просили ввести число Пи с некоторой точностью: строго заданное число знаков после запятой, не больше и не меньше. Достаточно точное число Пи можно взять на Википедии или, для пользователей Windows 10, Калькуляторе.

Отгадать необходимую точность можно с помощью метода проб и ошибок, или просто написав бота.

Ответ: `3.14159265358979323846264338327`

1.6 Задача, которую невозможно решить

Вопреки сказанному в названии задачи, ее возможно решить. Наличие решения этой задачи показывает, как внимательно участники прочитали инструкцию по работе с тестирующей системой. Ведь именно в разделе **Мануал**, в подразделе **Обработка персональных данных**, совершенно не к месту находилось предложение "Однако, для решения задачи, которую невозможно решить, вам нужно всего лишь отправить название конкурса". Остается лишь найти правильное с точностью до символа название.

Ответ: `Звёзды Кассиопеи`

1.7 Двадцать пятый кадр

В данной задаче участникам предлагался для анализа файл с видео. И хотя в метаданных на этот раз флага не было, там можно было обнаружить частоту кадров видео: 120 кадров в секунду. Это значит, что на экранах с частотой обновления 60 Гц и менее (а таких на момент проведения конкурсов подавляющее большинство) немалая часть кадров просто не будет отображаться. Но зачем же авторы дали видео с таким большим числом кадров, зная, что часть из них не будет показана? Неужели они хотели что-то там спрятать? Да. Смотрим видео покадрово и на 817 кадре замечаем серый текст на сером фоне.

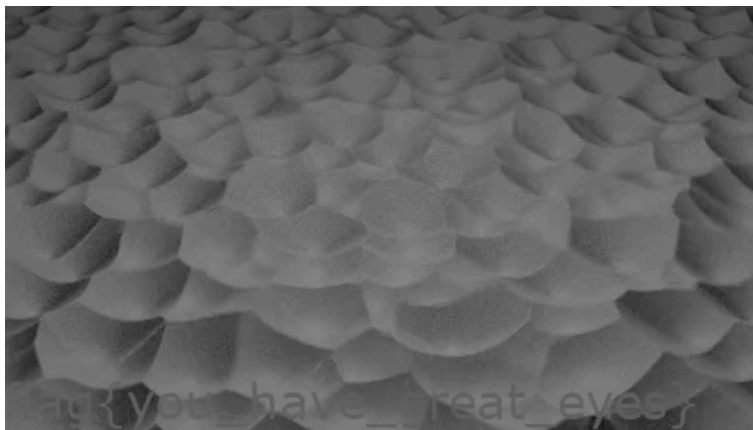


Рисунок 2: Один из кадров видео

Ответ: `flag{you_have_great_eyes}`

1.8 Englishman

В очередной раз отправляемся в Гугл (Яндекс, Bing, DuckDuckGo, Рамблер, Поиск Маил.Ру, ...) чтобы узнать, как сервер может узнать местоположение/страну/язык пользователя и узнаем, два основных способа: по IP и по предпочитаемому языку пользователя.

Методом проб и ошибок, испробовав все платные и бесплатные VPN-сервера узнаем, что в данной задаче IP ничего не значит.

Остается вариант с языком. Браузер внутри запроса отправляет информацию о предпочитаемом языке веб-страниц. В настройках браузера выставяем предпочтение на английский, отправляем что угодно в качестве ответа и получаем баллы.

1.9 1С

В следующей задаче на анализ содержимого файла участникам был дан файл Excel. Изучив свойства файла или создав новый лист, можно обнаружить, что кроме видимого листа с многозначным названием **Лист1** в книге есть еще два: **Лист2** и **Лист3**. В Microsoft Office Excel жмем ПКМ по списку листов, **Показать...**, **Лист3**. Изучаем содержимое ранее скрытого листа рядом с красивой картинкой аэродинамических характеристик коровы обнаруживаем флаг.

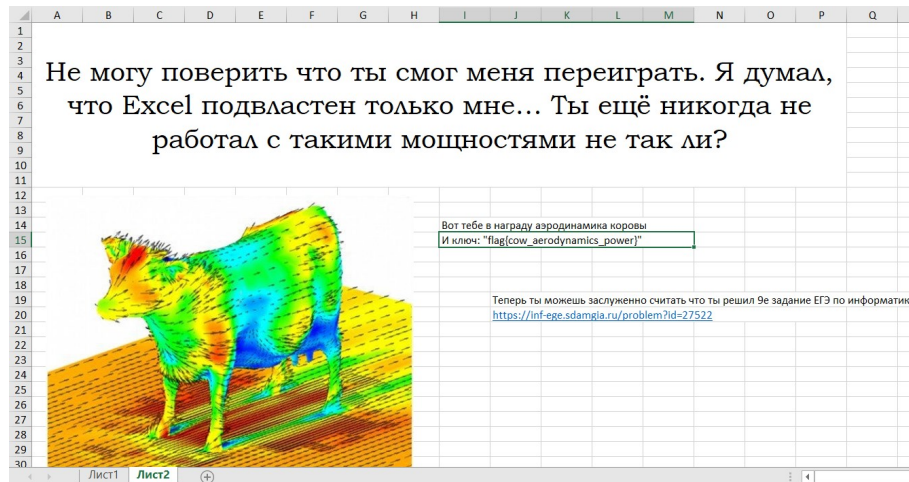


Рисунок 3: Скрытый лист с ответом

Ответ: `flag{cow_aerodynamics_power}`

1.10 Машина времени

Снова скачиваем приложенный файл. Распаковываем архив, наряду с привычными, но ничем не примечательными файлами обнаруживаем папку `.git`. Или не обнаруживаем, если у вас не включено отображение скрытых файлов. В таком случае, включаем, потому что без него заниматься поиском скрытой информации весьма непросто.

В любом случае, закинув название папки в Гугл получаем понимание, что авторы дали вам Git-репозиторий. Git - одна из систем контроля версий, то есть позволяет отслеживать и сохранять изменения файлов, откатываться к предыдущей версии.

Скачиваем Git, устанавливаем. Делаем коммит с помощью `git commit -a`, видим, что после последнего коммита (сохранения) файлы изменялись. Откатываемся к предыдущему коммиту `git checkout a23d2c7`, смотрим содержимое когда-то бывшего удаленным файла, находим флаг.

Ответ: `flag{wayback_machine_master}`

1.11 Восьмая симфония Чайковского

Любой уважающий себя третьеклассник знает, что самым лучшим форматом файла для хранения изображения является `mp3`. К пятому классу дети узнают, что это всё же не лучшее место для хранения картинок, но всё ещё подходящее. Я говорю об обложке альбома - картинке, которую можно вложить в файл `mp3`. При этом она не обязана действительно быть обложкой альбома, там может быть всё что угодно. Хотя, чтобы не вызывать много подозрений, мы нашли какую-то похожую картиночку, на которой написали флаг.



Рисунок 4: Картинка из файла 51.mp3

Ответ: `flag{fade_in_music}`

1.12 Я - Гуль

В данной задаче участникам предлагалось использовать обученную заранее нейронную сеть для выполнения задачи распознавания и анализа образов. С использованием такой нейронной сети можно с лёгкостью понять, какое математическое выражение находится в приложенном к заданию файле. Допускается также использование этой нейронной сети для вычисления значения выражения.

Если считать нейронной сетью множество связанных нейронов, то головной мозг человека можно назвать нейронной сетью. Я не биохим.

А задачу мы предлагаем решать вручную. То есть, нам лень писать нормальное решение.

1.13 Spectre

Название задачи говорит об амплитудно-частотном анализе. Это явно не просто так, в отличие от остальных задач. Заметим, что спектрограмма позволяет выполнить неплохой амплитудно-частотный анализ "на глаз".

Открываем файл в чем-нибудь, что умеет строить спектрограмму, например **Audacity**. Далее задача решается методом пристального взглядывания в эту спектрограмму.

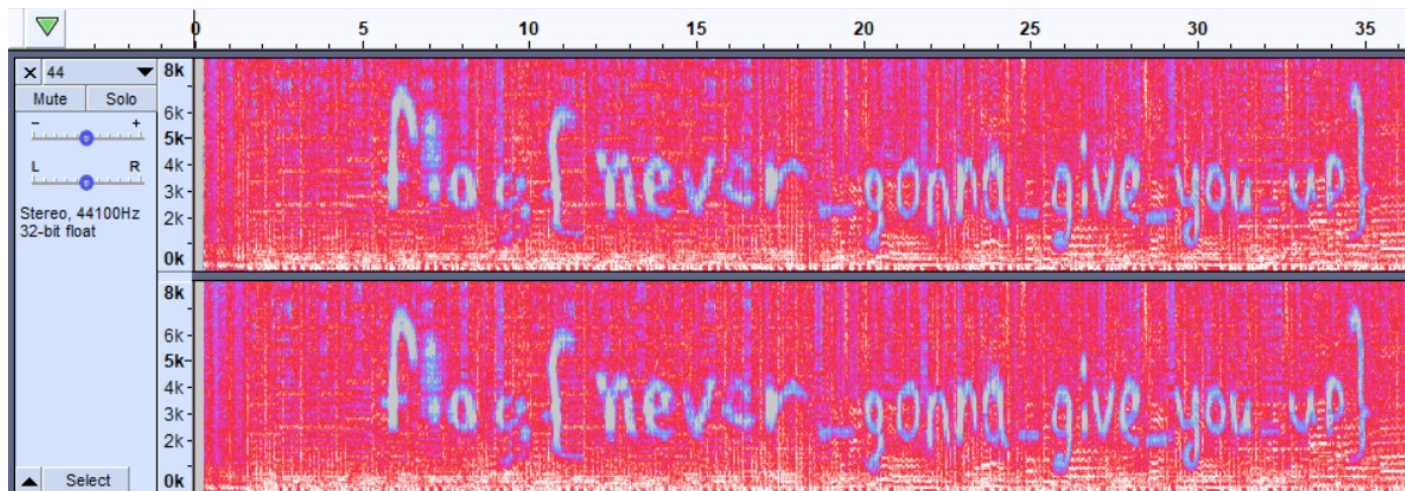


Рисунок 5: Фрагмент спектрограммы

Ответ: `flag{never_gonna_give_you_up}`

1.14 Пленочный фотоаппарат

Ответ: `flag{that_was_a_good_2.3BTC}`

1.15 Пленочный фотоаппарат 2

Ответ: `flag{that_was_my_last_money}`

2 Задачи на использование ботов

Для начала рассмотрим общую для всех ботов часть: работу с сетью. Мы использовали для написания ботов Python 3 и библиотеку `requests`. Ничто не мешало вам использовать другие варианты, а незнание любого из них компенсируется длиной конкурса и доступностью все того же Гугла.

Импортируем библиотеку и задаем константы:

```
# -*- coding: utf-8 -*- # Волшебное слово, чтобы использовать кириллицу
import requests # Импортируем requests

login = 'login' # Сохраняем логин и токен
token = '23847634320439b65adafb7e3fefff802b1cde0a1b2977c3fe299a6dd111e4ce'
host = 'https://fetefot763.eu.pythonanywhere.com/'
```

Фрагмент программы 1: Импортируем requests

Теперь нужно разобраться, как браузер отправляет запросы на сайт. Открываем панель разработчика и изучаем (смотри рисунок 6).

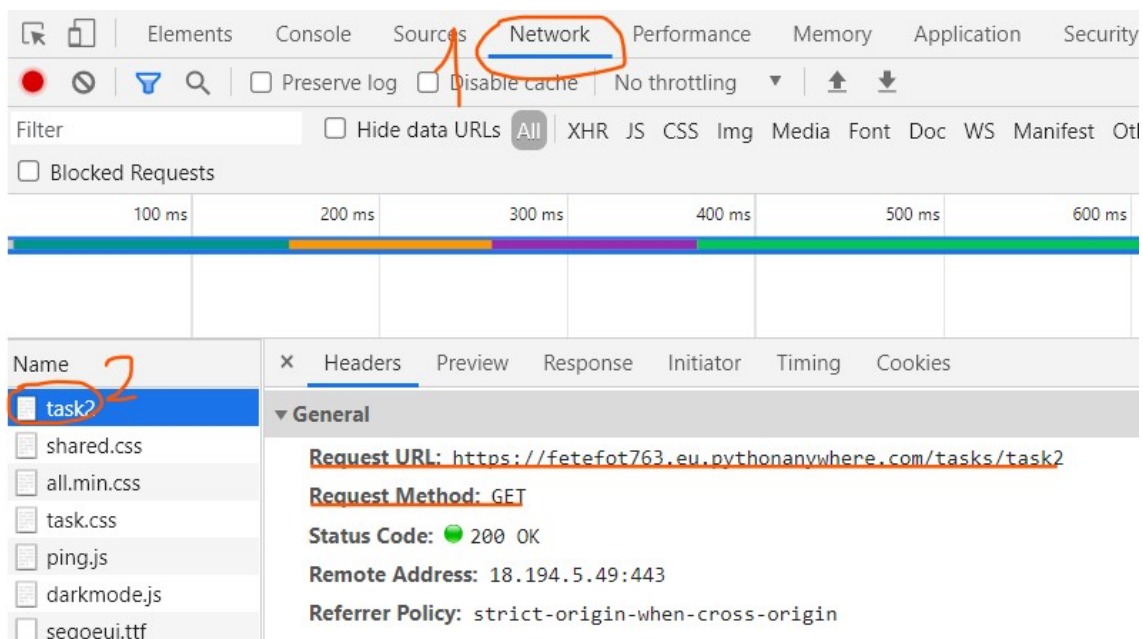


Рисунок 6: Изучение процесса получения страницы браузером.

Видим, что браузер отправляет GET-запрос на страницу задачи. Продолжаем изучение, проматываем вниз (смотри рисунок 7).

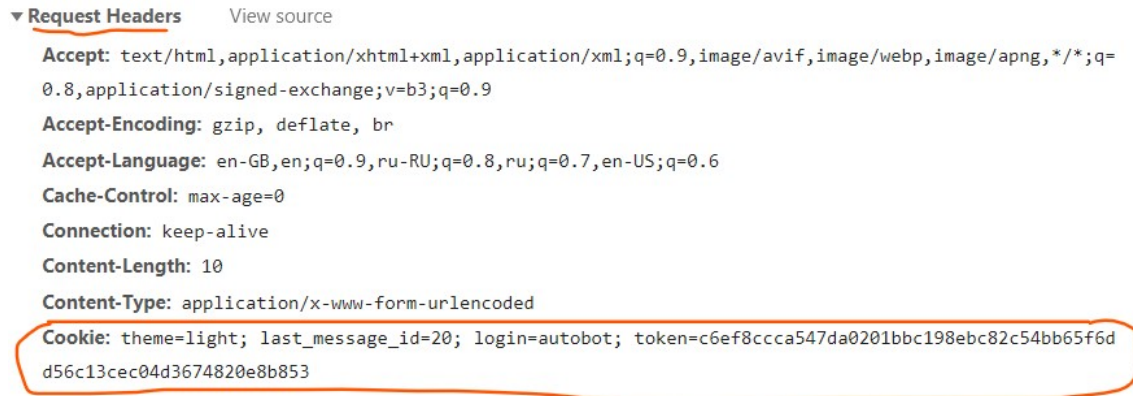


Рисунок 7: Изучение отправляемых браузером cookie-файлов.

Теперь видим, что браузер отправляет cookie-файлы, чтобы получить страницу. Почитав в мануале обнаруживаем, что для работы необходимо только две печеньки: `login` и `token`. Их значения сохраняем к себе и используем дальше.

Используя полученные знания мы можем написать функцию получения задачи. Функция - это обособленный фрагмент кода, выполняющий узкую задачу.

Для задач, где в приложении дан текст, мы можем получить этот текст прямо со страницы задачи:

```
# Для текстовых задач (на примере задачи №1 - Водолей)
def get_data():
    text = requests.get(
        host + 'tasks/task1',
        cookies={'login': login, "token": token}
    ).text
    return text
```

Объявляем функцию

Отправляем запрос

На страницу задачи

С cookies авторизации

И сохраняем текст ответа в переменную text

Возвращаем text

Фрагмент программы 2: Получение содержимого текстовой задачи

Заметим, что функция `get_data` возвращает нам весь HTML-код страницы. Нам же нужно только приложение к заданию. Приложения всегда находятся в блоке с `id="task_data"` и не содержат блоков внутри. Пользуясь этим, создадим новую функцию для извлечения этого приложения:

```
def get_text_from_data(data):
    marker_beg = '<div id="task-data">'
    marker_end = '</div>'
    ind_beg = data.rfind(marker_beg) + len(marker_beg) # Индекс начала содержимого блока в строке
    ind_end = data[ind_beg:].find(marker_end)           # Индекс конца содержимого блока в строке
    text = data[ind_beg:ind_beg + ind_end]              # Сохраняем содержимое блока в переменную
    return text                                         # Возвращаем text
```

Фрагмент программы 3: Получение текста задачи из содержимого

Следует четко понимать, какую задачу вы решаете и какие функции из приложенных вам следует использовать. Так, функция получения текста задачи вернет мусор, если текста в задаче не окажется. А что произойдет с вашей программой при попадании внутрь мусора никому не известно.

Если же в приложении дана ссылка на файл, то его нужно скачать. Тогда функция может ничего не возвращать: результат запроса уже сохранен в файл и будет прочитан из него.

```
# Для файловых задач (на примере задачи №2 - WinRar 3000)
def get_data(): # Объявляем функцию
    requests.get( # Отправляем запрос
        host + 'tasks/task2', # На страницу задачи
        cookies={'login': login, "token": token} # С cookies авторизации
    )
    with open('2.zip', 'wb') as file: # Создаем и открываем файл 2.zip для записи
        file.write( # Записываем в файл
            requests.get( # Отправляем запрос
                host + f'task-generated-content/2/task2_{team_id}.zip', # За файлом
                cookies={'login': login, "token": token} # С cookies авторизации
            ).content # Записываем в файл ответ на запрос
        )
```

Фрагмент программы 4: Получение приложенного файла задачи

Теперь проанализируем отправку браузером нашего решения задачи. Напишем что-нибудь в поле ответа и нажмем кнопку отправки. Посмотрим, что же нам показывает браузер в уже знакомой панели (смотри рисунок 8).

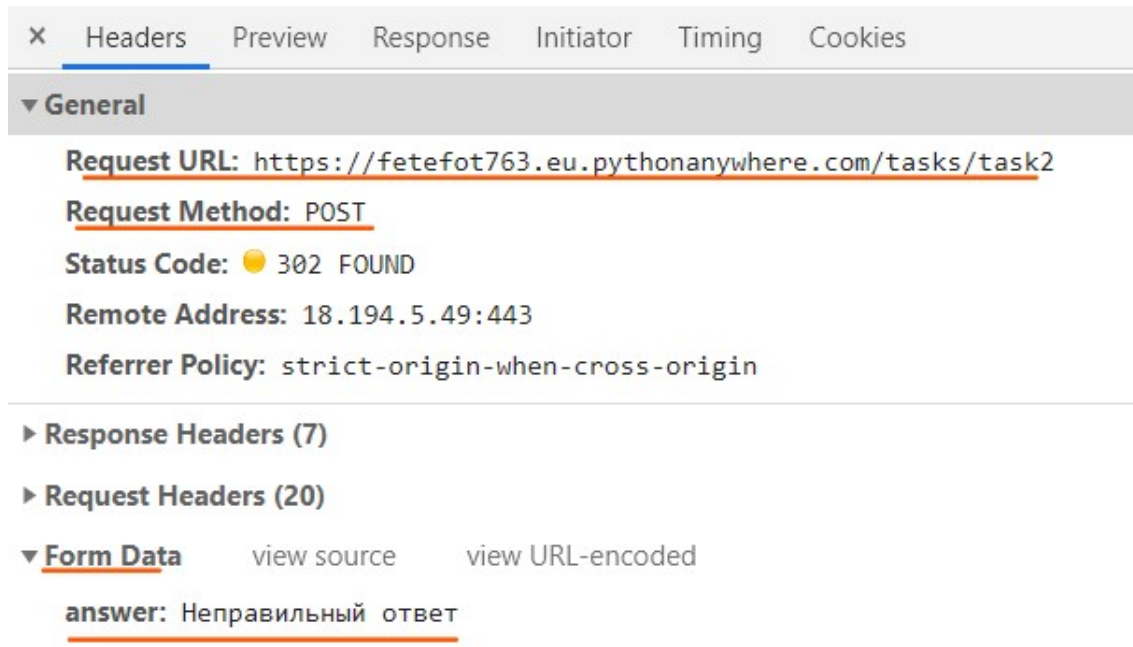


Рисунок 8: Изучение отправки браузером ответа на задачу.

Видим, что браузер отправляет запрос все на ту же страницу, но уже POST-запрос. В запрос браузер включает **данные формы**, содержащие поле **answer** со значением, которое мы только что вписали в поле ввода ответа. Как всегда, cookie браузер тоже отправляет.

Напишем функцию отправки ответа в данных формы:

```
def send_data(data):  
    requests.post(  
        host + f'tasks/task{task_id}',  
        data={'answer': data},  
        cookies={'login': login, "token": token}  
    )
```

Фрагмент программы 5: Отправка ответа на задачу

На этом работа с сетью завершается. Остается только в правильном порядке вызывать наши функции:

```
# Для текстовых задач  
if __name__ == '__main__':  
    while True:  
        txt = get_text_from_data(get_data())  
        res = solve(txt)  
        send_data(res)
```

Фрагмент программы 6: Основной цикл программы решения текстовой задачи

```
# Для файловых задач
if __name__ == '__main__':
    while True:
        get_data()
        res = solve()
        send_data(solve())
```

Если запущен именно этот файл
Вечный цикл
Получаем данные
Решаем задачу и сохраняем ответ
Отправляем ответ

Фрагмент программы 7: Основной цикл программы решения файловой задачи

Просто вставить этот код себе и запустить у вас не получится: функция `solve` не определена. Реализацию этой функции мы рассмотрим отдельно для каждой задачи.

2.1 Школьный конкурс компьютерной графики на тему "Программирование"

Здесь ничего решать не нужно, нужно просто отправить 29 апреля. Тогда функция `solve` может просто возвращать эту строку, без каких-либо вычислений.

```
def solve(text):  
    return '29 апреля'
```

Фрагмент программы 8: Функция `solve` для задачи ШККГНТП

2.2 Водолей

Решаем задачу как текстовую, но не с помощью этого кода, потому что он не работает.

```
def solve(text):  
    text = text.replace(',', ' ').replace('.', ' ') # Удаляем знаки препинания  
    text = text.replace('!', ' ').replace('?', ' ')  
    text = text.replace('-', ' ')  
    while '  ' in text:                               # Пока есть сдвоенные пробелы  
        text = text.replace('  ', ' ')                # Заменяем их одиночными  
    text = text.strip()                                # Удаляем пробелы по краям строки  
    res = text.split()                                 # Создаём список слов между пробелами  
    return len(res)                                   # Возвращаем длину этого списка
```

Фрагмент программы 9: Функция `solve` для задачи Водолей

2.3 WinRar 3000

Решаем задачу как файловую, скачиваем архив `2.zip`. Гуглим, как с помощью утилит командной строки на Windows распаковать архив, находим команду `tar -x -f 2.zip`. Гуглим, как с помощью Python запустить утилиту командной строки, находим библиотеку `subprocess`. Гуглим инструкции, соединяем, пишем код:

```
def solve():  
    subprocess.call(['tar', '-x', '-f', '2.zip']) # Вызываем утилиту распаковки архива  
    with open('answer.txt', 'r') as res:         # Открываем файл answer.txt на чтение  
        return res.read()                       # Возвращаем содержимое файла
```

Фрагмент программы 10: Функция `solve` для задачи WinRar 3000

2.4 Никита Егоров

Решаем задачу как текстовую. Вместо того, чтобы писать обработчик математики самому, воспользуемся плюсами Python и просто выполним строку с выражением. Полученный результат приведем к целому числу и отправим:

```
def solve(text):  
    return int(eval(text))
```

Фрагмент программы 11: Функция solve для задачи Никита Егоров

2.5 Emoji-Warrior

Решаем задачу как текстовую.

```
def solve(text):
```

2.6 Цветовод

Решаем задачу как файловую. Скачиваем картинку в 29.png, открываем, смотрим цвет и инвертируем его.

```
def solve():  
    img = Image.open('29.png') # Открываем файл с изображением  
    pix = img.load()           # Дооткрываем  
    return '#' + \              # Возвращаем '#' + ...  
        ''.join(               # ... соединенные пустой строкой элементы списка ...  
            map(                # ... в котором применим функцию к каждому элементу списка ...  
                lambda colour: hex(255 - colour)[2:].upper().zfill(2),  
                pix[0, 0]       # Функция инвертирования цвета и преобразования к нужному виду  
            )                  # Список с данными о цветах пикселя по каналам  
        )
```

Фрагмент программы 12: Функция solve для задачи Цветовод

2.7 Кадровое агенство

2.8 Вечеринка

В этой задаче нужно принести печеньеки с чаем. По понятным причинам, печеньека - это файл cookie. По уже не таким понятным, но все еще достижимым причинам, чай - это содержимое файла. Так, вместе с авторизационными печеньками отправляем cookie с названием и значением tea. На самом деле, достаточно и одного названия, или одного значения tea.

```
def send_data():
    text = requests.post(
        host + 'tasks/task21',
        cookies={'login': login, 'token': token, 'tea': 'tea'})
    return text
```

Фрагмент программы 13: Функция solve для задачи Вечеринка

2.9 Календарь

Решаем задачу как текстовую. Здесь нужно считать значение даты, добавить 701 один день и отправить обратно. Но ведь нужно считать все эти дни в месяце, високосные годы и другие сложные штуки. Но зачем, когда это уже сделано до нас?

Воспользуемся библиотекой `datetime` для расчёта даты.

```
import datetime

def solve(text):
    days = datetime.datetime.strptime(text, '%d.%m.%Y') \ # Волшебный перенос строки
        .date().toordinal()                               # Получаем число дней с начала времён
    days += 701                                           # Добавляем 701 день
    return datetime.date.fromordinal(days) \             # Волшебный перенос строки
        .strftime('%d.%m.%Y')                           # Возвращаем значение даты
```

Фрагмент программы 14: Функция solve для задачи Календарь

Однако авторы задачи столкнулись с некоторой особенностью работы библиотеки под разными ОС. В некоторых случаях год в дате был написан без лидирующих нулей, то есть не обязательно четырьмя символами.

2.10 Рукой подать

```
class Vector():
    x: int
    y: int

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def normalize(self):
        if self.x != 0: self.x = self.x // abs(self.x)
        if self.y != 0: self.y = self.y // abs(self.y)
```

```

def solve():
    img = Image.open('43.png')
    pix = img.load()
    x1, y1, x2, y2 = None, None, None, None

    for x in range(img.size[0]):
        for y in range(img.size[1]):
            if pix[x, y] == (0, 0, 0):
                if x1 is None:
                    x1 = x
                    y1 = y
                else:
                    x2 = x
                    y2 = y
    vec = Vector(x2 - x1, y2 - y1)
    vec.normalize()

    if vec.x == 1 and vec.y == 1: x1 += 1; y1 += 1
    if vec.x == -1 and vec.y == 1: y1 += 1; x2 += 1
    if vec.x == 1 and vec.y == -1: x1 += 1; y2 += 1
    if vec.x == -1 and vec.y == -1: x2 += 1; y2 += 1

    if vec.x == 0 and vec.y == 1: y1 += 1
    if vec.x == 0 and vec.y == -1: y2 += 1
    if vec.y == 0 and vec.x == 1: x1 += 1
    if vec.y == 0 and vec.x == -1: x2 += 1

    length = ((x2 - x1) ** 2 + (y2 - y1) ** 2) ** 0.5

    round_len = round(length, 5)
    if int(length) != length:
        ans = str(round_len)
    else:
        ans = str(int(round_len))
    return ans

```

2.11 CuSo4

Решаем задачу как текстовую.

Для начала, используя информацию из условия, подготовим список химических элементов с номерами. Скопируем первые столбцы таблицы из Википедии в Excel, далее экспортируем (сохраняем как) `.csv` - простой формат для программной работы с таблицами: строки разделяются символами перевода строки (внезапно), а столбцы - запятыми. Полученный файл сохраним как `chemlist.csv`.

```
1,H
2,He
3,Li
4,Be
5,B
```

Фрагмент программы 15: Фрагмент файла со списком химических элементов

Теперь необходимо в программе прочитать этот файл, после этого выполнить замены в правильном порядке.

```
with open('chemlist.txt', 'r') as chemlist:      # Открываем файл со списком элементов
    chem = [                                       # Создаем таблицу химических элементов
        i[:-1].lower().split(',')                # Разделяем строку на столбцы
        for i in chemlist.readlines()           # Из содержимого файла
    ]
chem.sort(key=lambda x: len(x[1]), reverse=True) # Сортируем по убыванию длины названия элемента

def solve(text):                                  # Объявляем функцию
    for elem in chem:                             # Для каждого химического элемента
        text = text.replace(elem[1], elem[0])    # Заменяем его название на номер
    return text                                    # Возвращаем текст
```

Фрагмент программы 16: Функция solve для задачи CuSo4