Разбор задач конкурса "Звезды Кассиопеи"

Надобных Дмитрий, Пугач Сергей, Чечеватов Роман

20.05.2021

1 Ручные задачи

1.1 MP5

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.

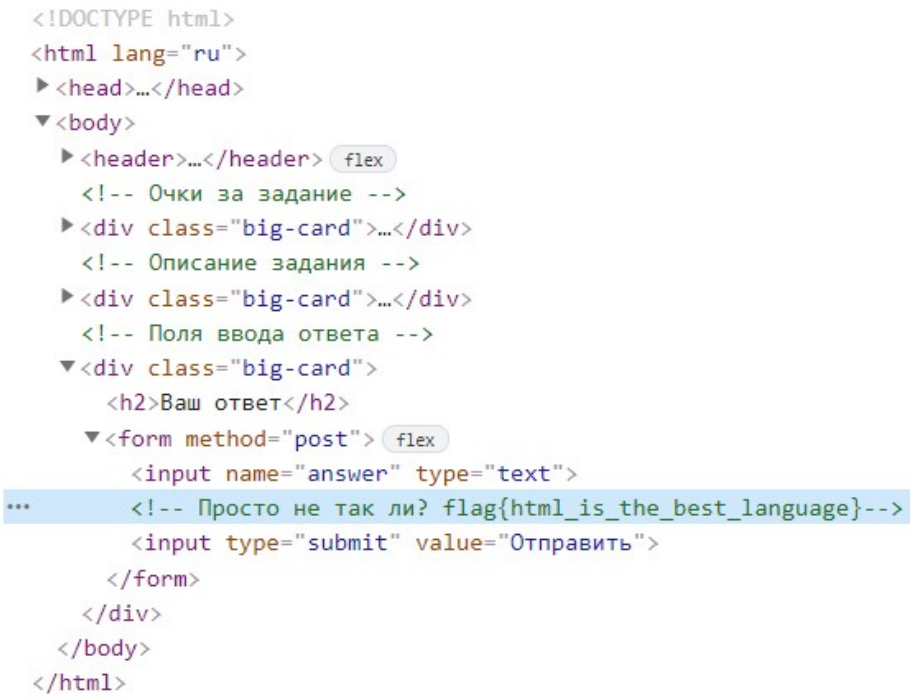

Рисунок 1: HTML-код страницы задачи

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.

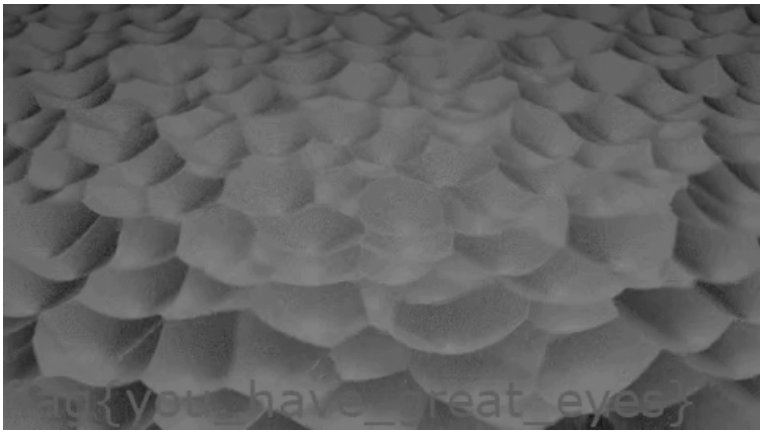

Рисунок 2: Один из кадров видео

Ответ: `flag{you_have_great_eyes}`

1.8 Eglishman

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 выставляем предпочтение на английский, отправляем что угодно в качестве ответа и получаем баллы.

1.9 1C

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 многозначащим названием `Лист1` в книге есть еще два: `Лист2` и `Лист3`. В Microsoft Office Excel жмем ПКМ по списку листов, `Показать...`, `Лист2`. Изучаем содержимое ранее скрытого листа рядом с красивой картинкой аэродинамических характеристик коровы обнаруживаем флаг.

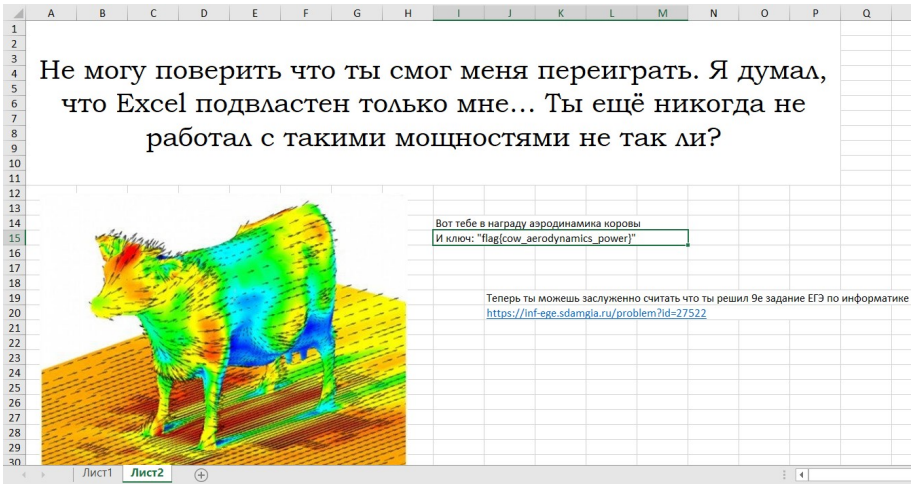

Рисунок 3: Скрытый лист с ответом

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. Да, писали светлым по светлому, но никто не обещал, что будет просто.

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. Серьёзно: чтобы получить образцы всех цифр, нужно правильно заслать не менее 8 раз. А после такого уже и до полного решения недалеко.

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 вглядывания в эту спектрограмму.

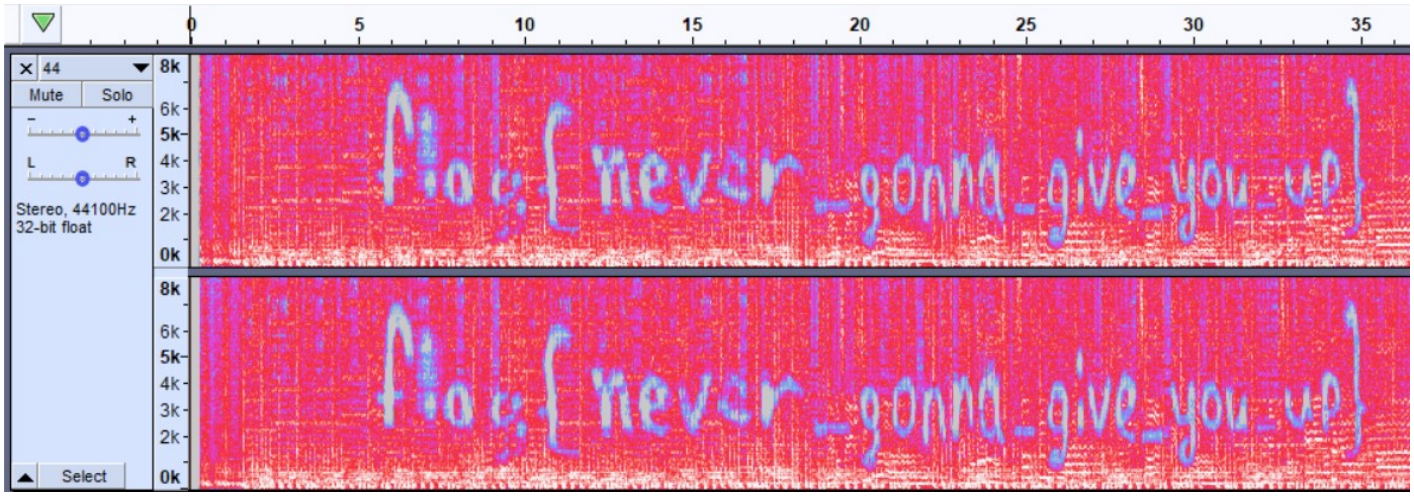

Рисунок 5: Фрагмент спектрограммы

Ответ: `flag{never_gonna_give_you_up}`

1.14 Пленочный фотоаппарат

Скачиваем предложенный файл и замечаем, что многие программы просмотра изображений отказываются её показывать. В детальных свойствах картинки видим, что изображение имеет размер 400000 x 1.

Ищем программу, которая откроет нам этот документ. Мы использовали лучший растровый редактор `Paint`. Открыв изображение видим, что оно состоит из белых и чёрных пикселей.

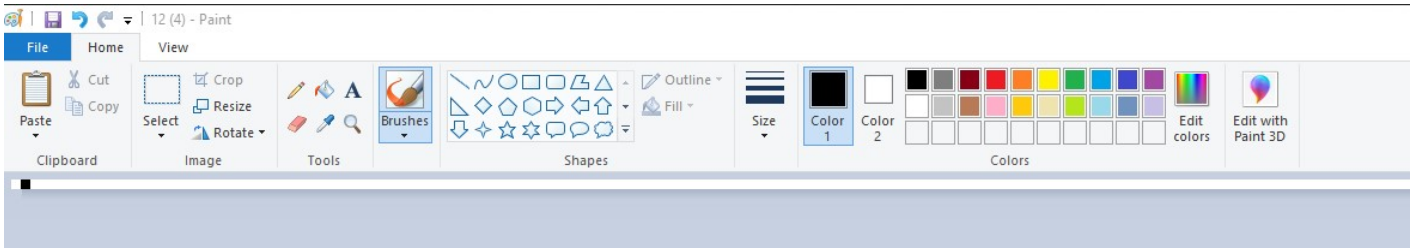

Рисунок 6: Вид файла в программе Paint

Методом пристального вглядывания узнаём, что в начале изображения чёрные пиксели повторяются каждые 400px. Заметим, что ширина изображения очень даже кратна 400 и понимаем, что перед нами картинка размером 400 x 1000, которую развернули в одну полосу, где каждый 400-й пиксель начинается следующая строка пикселей исходного изображения.

Решить задачу можно и вручную, разрезав изображение и растровом редакторе на 1000 полосок.

Но подразумевалось написание программы на языке `Python` с использованием библиотеки `Pillow` для преобразования исходного изображения в нужное.

```python
from PIL import Image, ImageDraw  # Импортируем библиотеку для работы с изображениями

filepath = '12.png'          # Путь до файла с картинкой
img = Image.open(filepath)  # Открываем картинку
# Выгружаем картинку в матрицу значениями которой является кортеж (R,G,B) значений
pix = img.load()
width, height = 400, 1000

# Создаём пустое изрбражение заданного размера, белого фона
new_img = Image.new("RGB", (width, height), (225, 225, 225))
# Создаём холст draw на котором будем рисовать пиксели в нужном порядке
draw = ImageDraw.Draw(new_img)
for y in range(height):
    for x in range(width):
        # Рисуем на холсте в координатах (x,y) пиксель, находящийся на 400 * y + x позиции
        draw.point((x,y), pix[width * y + x, 0])

new_img.save('solved2.png', "PNG")  # Сохраняем полученное изображение и радуемся
```

Ответ: `flag{that_was_a_good_2.3BTC}`

1.15 Пленочный фотоаппарат 2

Тут проделываем те же шаги, что и в первой версии задачи и получаем вместо читабельного флага, набор разбросанных чёрных пикселей.

Смотрим внимательно на изображение и видим, что помимо чёрных и белых пикселей, на изображении присутствуют красные.

Обращаем внимание, что на самый низ изображения красных пикселей не хватило, поэтому там их нет.

Красный пиксель есть в каждой строке, и он в ней ровно один. И там, где красные пиксели сгруппированы рядом, чёрные пиксели повторяют их положение

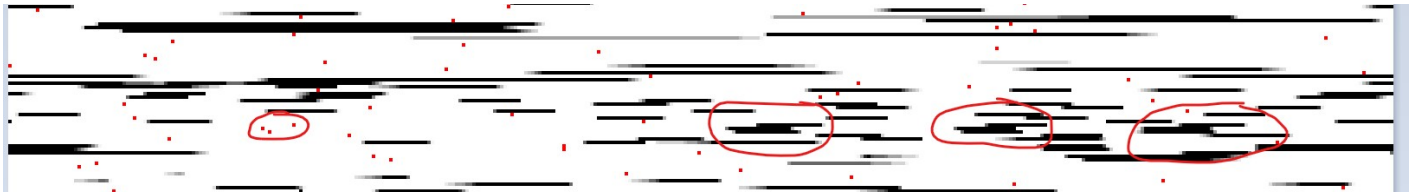

Рисунок 7: Вид файла в программе Paint

Каждая строка изображения циклически сдвинута на конкретное значение и красные пиксели нужны для того, чтобы однозначно определить значение сдвига и восстановить изображение. (Если вы не знаете, что такое «циклический сдвиг» загляните в интернет)

```
from PIL import Image, ImageDraw

filepath = '13.png'
img = Image.open(filepath)
pix = img.load()
width, height = 400, 1000

new_img = Image.new("RGB", (width, height), (225, 225, 225))
draw = ImageDraw.Draw(new_img)
for y in range(height):
    for x in range(width):
        draw.point((x,y), pix[width*y+x, 0])

# Код выше -- решение первого плёночного фотоаппарата

pix = new_img.load()        # Выгружаем полученную картинку в матрицу с цветами

# Заполняем нулями массив shift,
# в котором будем хранить положение красного пикселя для каждой строки
shift = [0 for i in range(height)]
for y in range(height):
    x = 0  # Перебираем x от 0 до 400, пока не встретим красный пиксель (255, 0, 0)
    while x < width and (pix[x, y][0] != 255 or pix[x, y][1] != 0 or pix[x, y][2] != 0):
        x += 1
    else:
        shift[y] = x - 1  # Встретив такой пиксель записываем x в shift

# Создаём новое изображение для рисования размером 400x1000
new_img2 = Image.new("RGB", (width, height), (225, 225, 225))
draw2 = ImageDraw.Draw(new_img2)
for y in range(height):
    for x in range(width):
        # Для каждой строки делаем циклический сдвиг на shift[i] где i номер строки
        draw2.point((x, y), pix[(x + shift[y]) % width, y])
```

```
new_img2.save("13-solution.png", "PNG")  # Сохраняем
```

Ответ: `flag{that_was_my_last_money}`

2 Задачи на использование ботов

Для начала рассмторим 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 8).

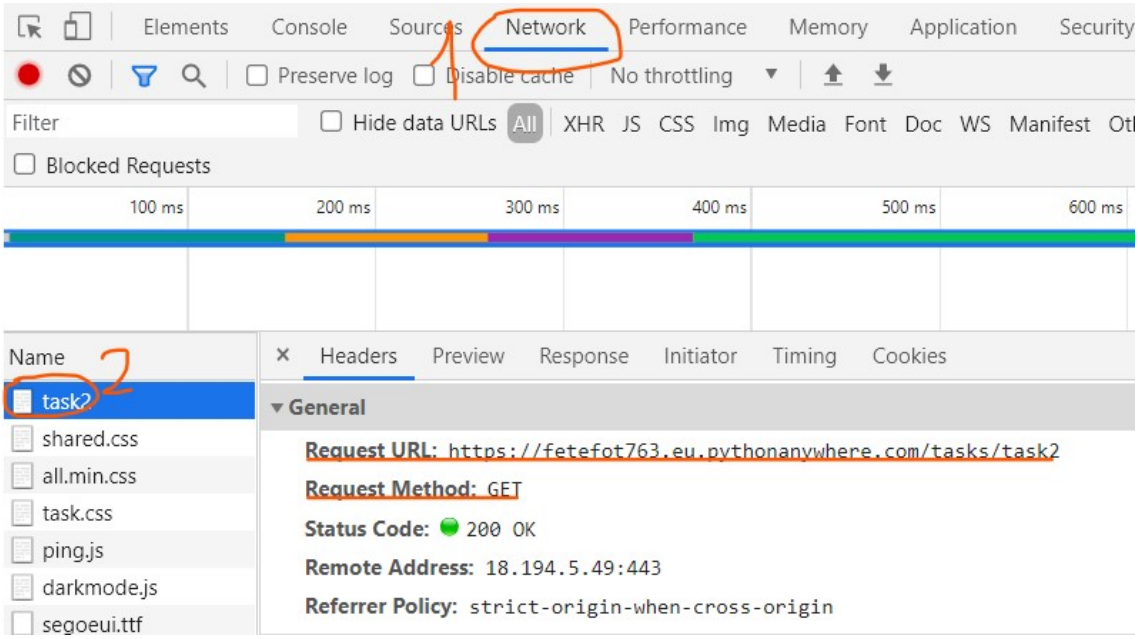

Рисунок 8: Изучение процесса получения страницы браузером.

Видим, что браузер отправляет GET-запрос на страницу задачи. Продолжаем изучение, проматываем вниз (смотри рисунок 9).

Рисунок 9: 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():                                     # Объявляем функцию
    text = requests.get(                            # Отправляем запрос
        host + 'tasks/task1',                       # На страницу задачи
        cookies={'login': login, "token": token}    # С cookies авторизации
    ).text                                          # И сохраняем текст ответа в переменную text
    return 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:

```python
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.

```python
# 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 10).

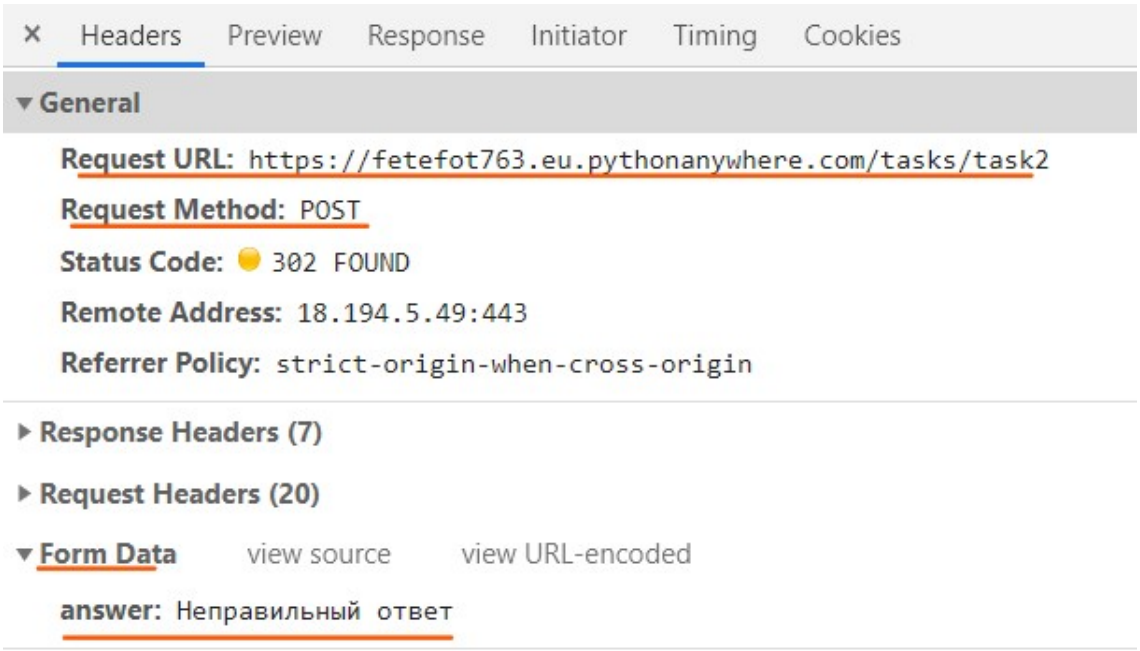

Рисунок 10: Изучение отправки браузером ответа на задачу.

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):                                # Объявляем функцию
    requests.post(                                  # Отправляем запрос
        host + f'tasks/task{task_id}',              # На страницу задачи
        data={'answer': data},                      # С данными формы
        cookies={'login': login, "token": token}    # И cookies авторизации
    )
```

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__':                          # Если запущен именно этот файл
    while True:                                     # Вечный цикл
        txt = get_text_from_data(get_data())        # Получаем текст задачи
        res = solve(txt)                            # Решаем задачу и сохраняем ответ
        send_data(res)                              # Отправляем ответ
```

Фрагмент программы 6: Основной цикл программы решения текстовой задачи

```
# Для файловых задач
if __name__ == '__main__':                    # Если запущен именно этот файл
    while True:                               # Вечный цикл
        get_data()                            # Получаем данные
        res = solve()                         # Решаем задачу и сохраняем ответ
        send_data(solve())                    # 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 рассмортим отдельно для каждой задачи.

2.1 Школьный конкурс компьютерной графики на тему "Программирование"

Здесь ничего решать не нужно, нужно просто отправить `29 апреля`. Тогда фукция `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. Получаем текст, заменяем знаки препинания на пробелы. Не убираем их, чтобы не соединять слова, а заменяем. Убираем все подряд идущие пробелы и пробелы по краям строки. Сичтаем слова.

```
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 одиночнымии
    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:

```python
import subprocess

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:

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

Фрагмент программы 11: Функция solve для задачи Никита Егоров

2.5 Emoji-Warrior

Не напрягаемся: берём, пишем правила замены вручную и заменяем. По техническим причинам будет только картинка. Да ещё и не со всем кодом, а только фрагмент

Рисунок 11: Функция solve для задачи Emoji-Warrior

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),
                                  # Функция инвертирования цвета и преобразования к нужному виду
                   pix[0, 0]      # Список с данными о цветах пикселя по каналам
               )
           )
```

Фрагмент программы 12: Функция solve для задачи Цветовод

2.7 Кадровое агенство

Для решения этой задачи желательно где-то раздобыть качественный список сотрудников школы. Отправляемся на сайт ШК, там в разделе `Сведения об образовательной организации` есть ссылка на `Руководство. Педагогический состав`. Переходим по ссылке, копируем список к себе в файл `.csv` - простой формат для программной работы с таблицами: строки разделяются символами перевода строки (внезапно), а столбцы - запятыми. Важно заметить и исправть опечатку в отчестве Кирюшиной Натальи: на сайте пропущена буква А.

```
staff = dict()                                          # Создаем пока пустой словарь
with open('14.csv') as file:                            # Открываем файл
    for line in file.readlines():                       # Читаем построчно
        man = line[:-1].split(',')                      # Получаем список с ФИО
        staff[' '.join([man[0], '...', man[2]])] = man[1] # Для строки Ф ... О ответ - Имя
        staff[' '.join([man[0], man[1], '...'])] = man[2] # Для строки Ф И ... ответ - Отчество

def solve(text):
    return staff[text]
```

Фрагмент программы 13: Функция solve для задачи Кадровое агенство

2.8 Вечеринка

В этой задаче нужно принести печеньки с чаем. По понятным причинам, печенька - 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 14: 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 15: 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 нолей, то есть не обязательно четырьмя символами.

2.10 Рукой подать

Условие в объяснении не требуется, используем библиотеку `Pillow`.

```
class Vector:               # Опишем класс вектора для удобства
    x: int
    y: int

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

    def simplize(self):     # Метод, где координаты вектора уменьшаем до 1, сохраняя знак
```

```
        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  # Координаты x y для первого и второго пикселей

    for x in range(img.size[0]):
        for y in range(img.size[1]):
            if pix[x, y] == (0, 0, 0):
                if x1 == None:      # Находим координаты первого чёрного пикселя
                    x1 = x
                    y1 = y
                else:               # Находим координаты второго чёрного пикселя
                    x2 = x
                    y2 = y
    vec = Vector(x2 - x1, y2 - y1)  # Создаём вектор с началом в первом пикселе и концов во втором
    vec.simplize()                  # Упрощаем его до вектора с координатами только 1, -1, или 0

    """
    Т.к. у нас записаны координаты только верхних левых углов пикселя, нам нужно определить
    по расположению пикселей, какие углы нужны для вычисления расстояния (См. условие)
    В зависимости от четверти, в которой лежит вектор, меняем x1, y1, x2, y2
    """
    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

    # Рассмотрим частные случаи, когда пиксели расположены на одной высоте или ровно друг под другом
    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

    # Вычисляем искомое расстояние, через формулу расстояний между точками
    length = ((x2 - x1)**2 + (y2 - y1)**2) ** 0.5
    round_len = round(length, 5)    # Округляем до 5 знаков
    if int(length) != length:       # Отбрасываем ".0", если он присутствует
        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 16: 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 17: Функция solve для задачи CuSo4