

## Проект API. Первый опыт по работе с Алисой

### План урока

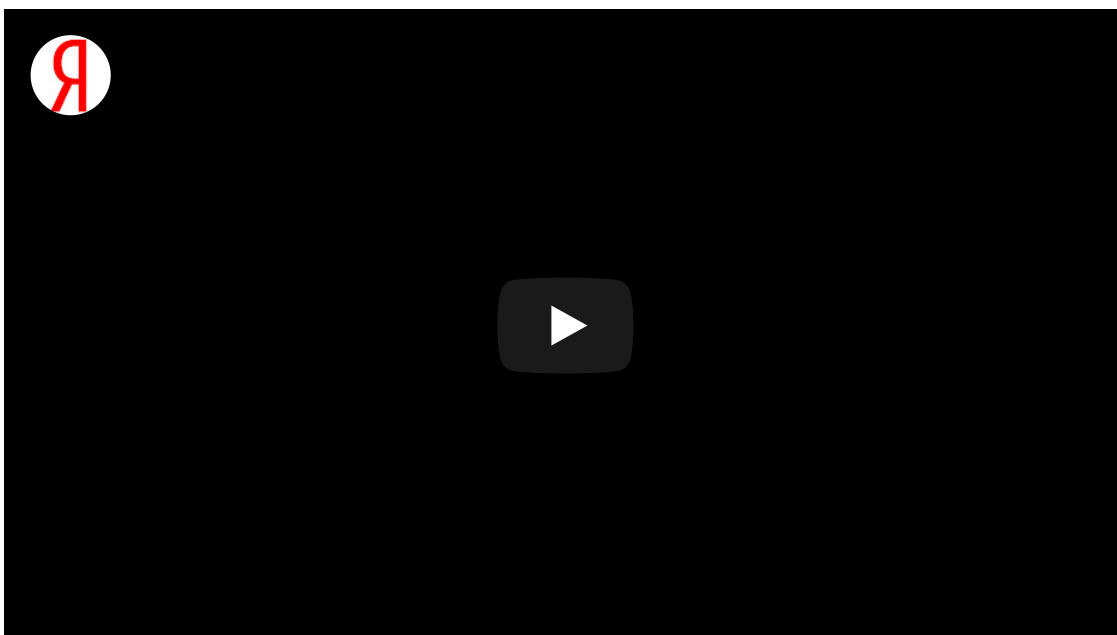
- 1 Алиса
- 2 Интеграция с Алисой
- 3 Логирование (журналирование)
- 4 Уровни логирования
- 5 Пишем код
- 6 Отладка с Postman
- 7 Pythonanywhere — запускаем наше приложение
- 8 Регистрация и тестирование навыка

### Аннотация

Мы начинаем блок по работе с голосовым помощником от Яндекса — Алисой.

## 1. Алиса

Алиса — это голосовой помощник от Яндекса. Фактически это робот, который умеет общаться с нами посредством компьютера или любого другого вычислительного устройства: мобильного телефона, часов или даже «умной колонки». В процессе общения Алиса может отвечать на вопросы, искать информацию в интернете, играть с нами в различные игры, торговать на бирже, информировать о погоде, распознавать образы и делать множество других вещей.



Навыками принято называть отдельные задачи, которые Алиса умеет решать. Например, программист может научить Алису заказывать пиццу. Это умение и будет навыком. Сейчас различными разработчиками создано большое количество навыков, мы же с вами сегодня сделаем простой навык под названием «Купи слона», в котором Алиса будет с нами играть в простую игру:

- Купи слона
- Нет
- Все говорят «Нет». А ты купи слона
- Хорошо
- Слона можно найти на Яндекс.Маркете!

И на этом всё. Эта игра — пример, который приведён на официальном сайте платформы Яндекс.Диалоги — места, где любой человек может публиковать свои навыки для Алисы.

Но для начала работы узнаем несколько вещей о разработке навыков.

## 2. Интеграция с Алисой

Первое, в чём нужно разобраться, — это как интегрироваться с Алисой. Алиса не имеет API в классическом понимании — как, например, у Яндекс.Карт. Интеграция происходит по технологии **WebHook** (технология — это громкое слово, скорее, идея :)).

Почему Алиса просто не могла предоставить нам API для работы? Как вы думаете, в чём сложность?

Алиса может обратиться к нашему навыку в любое время, поскольку никто не знает, когда один из миллионов пользователей Алисы решит им воспользоваться. Если бы Алиса давала своё API, то мы должны были бы регулярно её спрашивать: «Эй, Алиса, никто не запрашивал наш навык?», затем при положительном ответе: «Запрашивали? Вот мой ответ. А что там ответил пользователь?» и так далее.

Как же быть?

Оказывается, лучшим решением является такое, в котором Алиса сама сообщит нам обо всех событиях, которые нас касаются. Идея **WebHook'ов** состоит в том, что мы не обращаемся к API, а реализуем своё, но по правилам, описанным в документации. Даём доступ к созданному API Алисе, и она уже начинает общаться с ним самостоятельно. Получается как бы «интеграция наоборот». В документации Алисы есть строгие требования, как мы должны реализовать API. После того как API реализован, мы «говорим» Алисе, куда обращаться, — сообщаем наш адрес. Если в интерфейсе Алисы кто-то вызвал наш навык, то Алиса сама нам об этом сообщит. Очень удобно! Не надо постоянно «пинать» Алису.

## 3. Логирование (журналирование)

Представьте ситуацию, когда вы не можете видеть на мониторе работу вашей программы. У вас просто может не быть монитора. Или программа размещена не на вашем компьютере, а где-то в облаке. И вы понимаете, что программа работает некорректно. Как же узнать, как она работает и где вкралась ошибка? В таком случае помогает логирование.

**Логирование** — это запись и вывод информации о выполняемых операциях, ошибках и других событиях в коде. Это могут быть значения переменных или любой текст. Логирование может происходить в консоль или файл. Логи выводятся в консоль, например, в процессе отладки приложения. Если вы уже написали программу, которая выполняется на сервере, то записывать логи желательно в файл, чтобы потом их можно было изучить в случае остановки, падения или перезагрузки сервера.

Для логирования в Python используется библиотека **logging**.

Пример:

```
import logging

def log():
    i = 0
    while i < 10:
        logging.warning(i)
        i += 1

if __name__ == '__main__':
    log()
```

Выводит в консоль:

```
WARNING:root:0
WARNING:root:1
WARNING:root:2
WARNING:root:3
WARNING:root:4
WARNING:root:5
WARNING:root:6
WARNING:root:7
WARNING:root:8
WARNING:root:9
```

В этом примере мы используем библиотеку **logging** с параметрами по умолчанию, поэтому вся информация выводится в консоль. Мы использовали функцию `warning()`, которая предназначена для привлечения внимания к проблеме, которая ещё не считается ошибкой. **root** — это имя журнала, его можно менять.

Давайте научимся сохранять данные журнала в файл. Ещё раз напомним, что это — правильная практика, когда программа используется в боевом режиме (production), так как файл сохранится в случае остановки, падения или перезагрузки сервера.

Для логирования в файл нужно добавить в наш код следующую строку:

```
logging.basicConfig(filename='example.log')
```

Теперь код целиком выглядит так:

```
import logging
```

```
logging.basicConfig(filename='example.log')
```

```
def log_to_file():  
    i = 0  
    while i < 10:  
        logging.warning(i)  
        i += 1  
  
if __name__ == '__main__':  
    log_to_file()
```

После того как мы запустим программу, в папке программы создастся файл **example.log**, и в нём окажется та же информация, что ранее выводилась в консоль:

```
WARNING:root:0  
WARNING:root:1  
WARNING:root:2  
WARNING:root:3  
WARNING:root:4  
WARNING:root:5  
WARNING:root:6  
WARNING:root:7  
WARNING:root:8  
WARNING:root:9
```

Если мы запустим программу повторно, то увидим, что старые данные из файла не удаляются. Таким образом, информация будет записана дважды.

Чего же не хватает в файле? Что ещё нам важно знать? Правильно — время события, ведь без него никакого расследования не получится.

Для того, чтобы вывести временную метку, надо в функцию `logging.basicConfig()` передать ещё один параметр **format**. Как несложно догадаться, это формат сообщения (как выглядит и что содержит), которое записывается в лог. Сообщение в логе может содержать много полезной информации (подробно можно изучить [тут](#)). Мы рассмотрим только четыре атрибута, которые можно вывести, и подробно расскажем о них.

С добавлением параметра **format** наш код будет выглядеть так:

```
import logging  
  
logging.basicConfig(filename='example.log',  
                    format='%(asctime)s %(levelname)s %(name)s %(message)s')
```

```
def log_to_file():
    i = 0
    while i < 10:
        logging.warning(i)
        i += 1

if __name__ == '__main__':
    log_to_file()
```

Рассмотрим параметр `format='%(asctime)s %(levelname)s %(name)s %(message)s'`. Здесь:

<b>asctime</b>	Время события
<b>levelname</b>	Уровень логирования (подробно об этом расскажем ниже)
<b>name</b>	Имя логера (журнала). По умолчанию <b>root</b> , но вы можете задать разные имена, чтобы сделать свои логи более информативными. Например, считается хорошей практикой назначать имя файла с кодом
<b>message</b>	Сообщение, которые вы отправили в <code>logging.warning()</code>

После запуска мы увидим в файле:

```
2018-12-23 21:08:58,805 WARNING root 0
2018-12-23 21:08:58,805 WARNING root 1
2018-12-23 21:08:58,805 WARNING root 2
2018-12-23 21:08:58,805 WARNING root 3
2018-12-23 21:08:58,805 WARNING root 4
2018-12-23 21:08:58,806 WARNING root 5
2018-12-23 21:08:58,806 WARNING root 6
2018-12-23 21:08:58,806 WARNING root 7
2018-12-23 21:08:58,806 WARNING root 8
2018-12-23 21:08:58,806 WARNING root 9
```

## 4. Уровни логирования

Запись в логи принято разбивать на типы — уровни. Любое сообщение несёт в себе информацию определённой важности, и время реакции на сообщения отличается. Приведём примеры:

---

Уровень	Пример события
<b>Debug</b>	Отправлен запрос в базу данных на сохранение
<b>Debug</b>	Завершён запрос в базу данных на сохранение
<b>Debug</b>	Запрос в базу занял 0.02 секунды, извлечено 1000 записей
<b>Info</b>	Зарегистрирован новый пользователь (user_id = 123123)
<b>Warn</b>	Отклонена транзакция с суммой платежа 0
<b>Error</b>	Ошибка при сохранении данных пользователя (user_id = 123124, operation_id = 12312313)
<b>Critical (Fatal)</b>	Ошибка ответа API Яндекс Карт, код: 404. Маршруты не рассчитываются.

Начинающему программисту бывает сложно определить, к какому уровню следует отнести то или иное событие, возникающее при работе программы. Мы приведём базовые принципы, однако надо иметь в виду, что правила всегда определяет руководитель команды, которая работает над проектом.

Уровень	Относящиеся события
<b>Debug:</b>	Сообщения отладки. В боевом режиме (production) сообщения этого уровня обычно отключены, чтобы не засорять файлы журналов. Но они могут включаться для поиска багов, которые не удалось воспроизвести.
<b>Info:</b>	Обычные сообщения, информирующие о действиях программы. Реагировать на такие сообщения вообще не надо, но они могут помочь, например, при поиске багов, расследовании интересных ситуаций и т.д.
<b>Warn:</b>	Записывая такое сообщение, программа обычно пытается привлечь внимание. Произошло что-то странное. Возможно, это новый тип ситуации, ещё не известный на текущий момент. Следует разобраться в том, что произошло, что это означает, и отнести ситуацию либо к инфо-сообщению, либо к ошибке. Соответственно, придётся доработать код обработки таких ситуаций.
<b>Error:</b>	Ошибка в работе программы, требующая вмешательства. Что-то не сохранилось, что-то отвалилось. Необходимо принимать меры довольно быстро! Ошибки этого уровня и выше требуют немедленной записи в лог, чтобы ускорить реакцию на них.
<b>Critical (Fatal):</b>	Это — особый класс ошибок, приводящих к неработоспособности программы в целом или неработоспособности одного из её модулей. Чаще всего случаются фатальные ошибки из-за неверной конфигурации или отказов оборудования. Требуют срочной, немедленной реакции.

Имейте в виду, что это рекомендации. Вы как архитектор программы вправе определять критичность события или ошибки в своей программе, а если работа происходит в команде,

то решение о критичности той или иной ситуации принимается руководителем команды или коллегиально.

Разработчик может настраивать уровень логирования. Логируются сообщения установленного уровня и уровней более высокой критичности. Например, если установлен уровень **Info**, то в консоль или в файл будут выводиться сообщения уровней: **Info**, **Warn**, **Error** и **Fatal**.

Уровень логирования по умолчанию — **WARNING**.



Уровень логирования в Python настраивается всё в той же функции `logging.basicConfig()` следующим образом:

```
import logging

logging.basicConfig(level=logging.DEBUG)

def log():

    logging.debug('Debug')
    logging.info('Info')
    logging.warning('Warning')
    logging.error('Error')
    logging.critical('Critical or Fatal')

if __name__ == '__main__':
    log()
```

Видим в консоли:

```
DEBUG:root:Debug
INFO:root:Info
```



```
WARNING:root:Warning
ERROR:root:Error
CRITICAL:root:Critical or Fatal
```

Если вызвать `logging.basicConfig(level=logging.ERROR)`, то в консоли мы увидим:

```
ERROR:root:Error
CRITICAL:root:Critical or Fatal
```

Используйте логирование в процессе разработки. Это поможет вам правильно и своевременно реагировать на ошибки, искать баги и понимать, что происходит в вашей программе в любой момент времени.

## 5. Пишем код

Теперь переходим к самому интересному. Начнём! Мы разработаем навык, который описан в разделе «Быстрый старт» документации Алисы.

Вы уже знакомы с библиотекой **Flask**. С её помощью мы напишем небольшой веб-сервер, который будет обрабатывать запросы от Алисы.

Алиса будет передавать нам JSON, содержащий данные о пользователе, и информацию, которую пользователь ввёл, мы же должны будем вернуть в ответ свой JSON (соответственно документации). Алиса его обработает и покажет пользователю.

Рассмотрим JSON запроса, который отправит нам Алиса. На самом деле JSON выглядит сложнее и имеет больше полей, но мы рассмотрим только те из них, которые будем использовать (внимательно изучить содержание JSON'ов можно в документации):

```
{
  "request": {
    "command": "закажи пиццу на улицу льва толстого, 16 на завтра",
    "original_utterance": "закажи пиццу на улицу льва толстого, 16 на завтра"
  },
  "session": {
    "new": true,
    "message_id": 4,
    "session_id": "2eac4854-fce721f3-b845abba-20d60",
    "skill_id": "3ad36498-f5rd-4079-a14b-788652932056",
    "user_id": "AC9WC3DF6FCE052E45A4566A48E6B7193774B84814CE49A922E163B8B29881DC"
  },
}
```

```
"version": "1.0"
}
```

**request** — данные, полученные от пользователя.

**command** — запрос, который был передан вместе с командой активации навыка. Например, если пользователь активирует навык словами «спроси у Сбербанка, где ближайшее отделение», в этом поле будет передана строка «где ближайшее отделение».

**original\_utterance** — полный текст пользовательского запроса, максимум 1024 символа.

**session** — данные о сессии (разговоре с Алисой).

**new** — признак новой сессии. Возможные значения: true — пользователь начинает новый разговор с навыком, false — запрос отправлен в рамках уже начатого разговора.

**message\_id** — идентификатор сообщения в рамках сессии, максимум 8 символов. Инкрементируется (увеличивается на единицу) с каждым следующим запросом.

**session\_id** — уникальный идентификатор сессии, максимум 64 символа.

**skill\_id** — идентификатор вызываемого навыка, присвоенный при создании.

**user\_id** — идентификатор экземпляра приложения, в котором пользователь общается с Алисой, максимум 64 символа. Даже если пользователь авторизован с одним и тем же аккаунтом в приложении Яндекс для Android и iOS, Яндекс.Диалоги (так называется сервис Яндекса, управляющий навыками для Алисы) присвоят отдельный user\_id каждому из этих приложений.

**version** — версия протокола. Текущая версия — 1.0.

JSON ответа Алисы (то есть тот JSON, что сформирует наша программа) будет выглядеть так (конечно же, полное описание можно посмотреть в [документации](#)):

```
{
  "response": {
    "text": "Здравствуйте! Это мы, хороводоведы.",
    "tts": "Здравствуйте! Это мы, хоров+одо в+еды.",
    "buttons": [
      {
        "title": "Надпись на кнопке",
        "payload": {},
        "url": "https://example.com/",
        "hide": true
      }
    ],
    "end_session": false
  },
  "session": {
```

```
"session_id": "2eac4854-fce721f3-b845abba-20d60",
"message_id": 4,
"user_id": "AC9WC3DF6FCE052E45A4566A48E6B7193774B84814CE49A922E163B8B29881DC"
},
"version": "1.0"
}
```

**response** — данные для ответа пользователю.

**text** — текст, который следует показать и сказать пользователю. Максимум 1024 символа. Не должен быть пустым.

**buttons** — массив кнопок, которые следует показать пользователю. Все указанные кнопки выводятся после основного ответа Алисы, описанного в свойстве text. Кнопки можно использовать как релевантные ответу ссылки или подсказки для продолжения разговора.

**title** — текст кнопки, обязателен для каждой кнопки. Максимум 64 символа.

**url** — URL, который должна открывать кнопка, максимум 1024 байта.

**hide** — признак того, что кнопку нужно убрать после следующей реплики пользователя. Допустимые значения: false — кнопка должна оставаться активной (значение по умолчанию), true — кнопку нужно скрывать после нажатия.

**end\_session** — признак конца разговора. Допустимые значения: false — сессию следует продолжить, true — сессию следует завершить.

**session** — данные о сессии (аналогичные запросу).

**version** — версия протокола. Текущая версия — 1.0.

Вы можете скачать [файл с шаблоном кода](#), или скопировать его ниже.

```
# импортируем библиотеки
from flask import Flask, request
import logging

# библиотека, которая нам понадобится для работы с JSON
import json

# создаём приложение
# мы передаём __name__, в нём содержится информация,
# в каком модуле мы находимся.
# В данном случае там содержится '__main__',
# так как мы обращаемся к переменной из запущенного модуля.
# если бы такое обращение, например, произошло внутри модуля logging,
# то мы бы получили 'logging'
app = Flask(__name__)
```

```

# Устанавливаем уровень логирования
logging.basicConfig(level=logging.INFO)

# Создадим словарь, чтобы для каждой сессии общения с навыком хранились
# подсказки, которые видел пользователь.
# Это поможет нам немного разнообразить подсказки ответов
# (buttons в JSON ответа).
# Когда новый пользователь напишет нашему навыку, то мы сохраним
# в этот словарь запись формата
# sessionStorage[user_id] = {'suggests': ["Не хочу.", "Не буду.", "Отстань!"]}
# Такая запись говорит, что мы показали пользователю эти три подсказки.
# Когда он откажется купить слона,
# то мы уберем одну подсказку. Как будто что-то меняется :)
sessionStorage = {}

@app.route('/post', methods=['POST'])
# Функция получает тело запроса и возвращает ответ.
# Внутри функции доступен request.json - это JSON,
# который отправила нам Алиса в запросе POST
def main():
    logging.info('Request: %r', request.json)

    # Начинаем формировать ответ, согласно документации
    # мы собираем словарь, который потом при помощи библиотеки json
    # преобразуем в JSON и отдадим Алисе
    response = {
        'session': request.json['session'],
        'version': request.json['version'],
        'response': {
            'end_session': False
        }
    }

    # Отправляем request.json и response в функцию handle_dialog.
    # Она сформирует оставшиеся поля JSON, которые отвечают
    # непосредственно за ведение диалога
    handle_dialog(request.json, response)

    logging.info('Response: %r', request.json)

    # Преобразовываем в JSON и возвращаем
    return json.dumps(response)

def handle_dialog(req, res):

```

```

user_id = req['session']['user_id']

if req['session']['new']:
    # Это новый пользователь.
    # Инициализируем сессию и поприветствуем его.
    # Запишем подсказки, которые мы ему покажем в первый раз

    sessionStorage[user_id] = {
        'suggests': [
            "Не хочу.",
            "Не буду.",
            "Отстань!",
        ]
    }
    # Заполняем текст ответа
    res['response']['text'] = 'Привет! Купи слона!'
    # Получим подсказки
    res['response']['buttons'] = get_suggests(user_id)
    return

# Сюда дойдем только, если пользователь не новый,
# и разговор с Алисой уже был начат
# Обрабатываем ответ пользователя.
# В req['request']['original_utterance'] лежит весь текст,
# что нам прислал пользователь
# Если он написал 'ладно', 'куплю', 'покупаю', 'хорошо',
# то мы считаем, что пользователь согласился.
# Подумайте, всё ли в этом фрагменте написано "красиво"?
if req['request']['original_utterance'].lower() in [
    'ладно',
    'куплю',
    'покупаю',
    'хорошо'
]:
    # Пользователь согласился, прощаемся.
    res['response']['text'] = 'Слона можно найти на Яндекс.Маркете!'
    res['response']['end_session'] = True
    return

# Если нет, то убеждаем его купить слона!
res['response']['text'] = 'Все говорят "%s", а ты купи слона!' % (
    req['request']['original_utterance']
)
res['response']['buttons'] = get_suggests(user_id)

```

*# Функция возвращает две подсказки для ответа.*

```

def get_suggests(user_id):
    session = sessionStorage[user_id]

    # Выбираем две первые подсказки из массива.
    suggests = [
        {'title': suggest, 'hide': True}
        for suggest in session['suggests'][:2]
    ]

    # Убираем первую подсказку, чтобы подсказки менялись каждый раз.
    session['suggests'] = session['suggests'][1:]
    sessionStorage[user_id] = session

    # Если осталась только одна подсказка, предлагаем подсказку
    # со ссылкой на Яндекс.Маркет.
    if len(suggests) < 2:
        suggests.append({
            "title": "Ладно",
            "url": "https://market.yandex.ru/search?text=слон",
            "hide": True
        })

    return suggests

if __name__ == '__main__':
    app.run()

```

Сохраните ваш файл под именем **flask\_app.py**.

Разберём тезисно, как работает приведённая программа.

1. Когда мы запускаем программу, точкой входа в неё со стороны Алисы является функция `main()`, которая «обёрнута» декоратором **`app.route`**.
2. Внутри `main()` мы сначала логируем полученный запрос, затем начинаем формировать ответ, потом вызываем функцию `handle_dialog()` для обработки диалога с пользователем, в результате же логируем и возвращаем ответ.
3. Функция `handle_dialog()` формируем ответ, исходя из данных запроса и состояния сессии (новый разговор, или продолжение старого).
4. Вспомогательная функция `get_suggests()` формирует подсказки в ответе.
5. В глобальном словаре **`sessionStorage`** мы будем хранить информацию о подсказках для каждого пользователя.

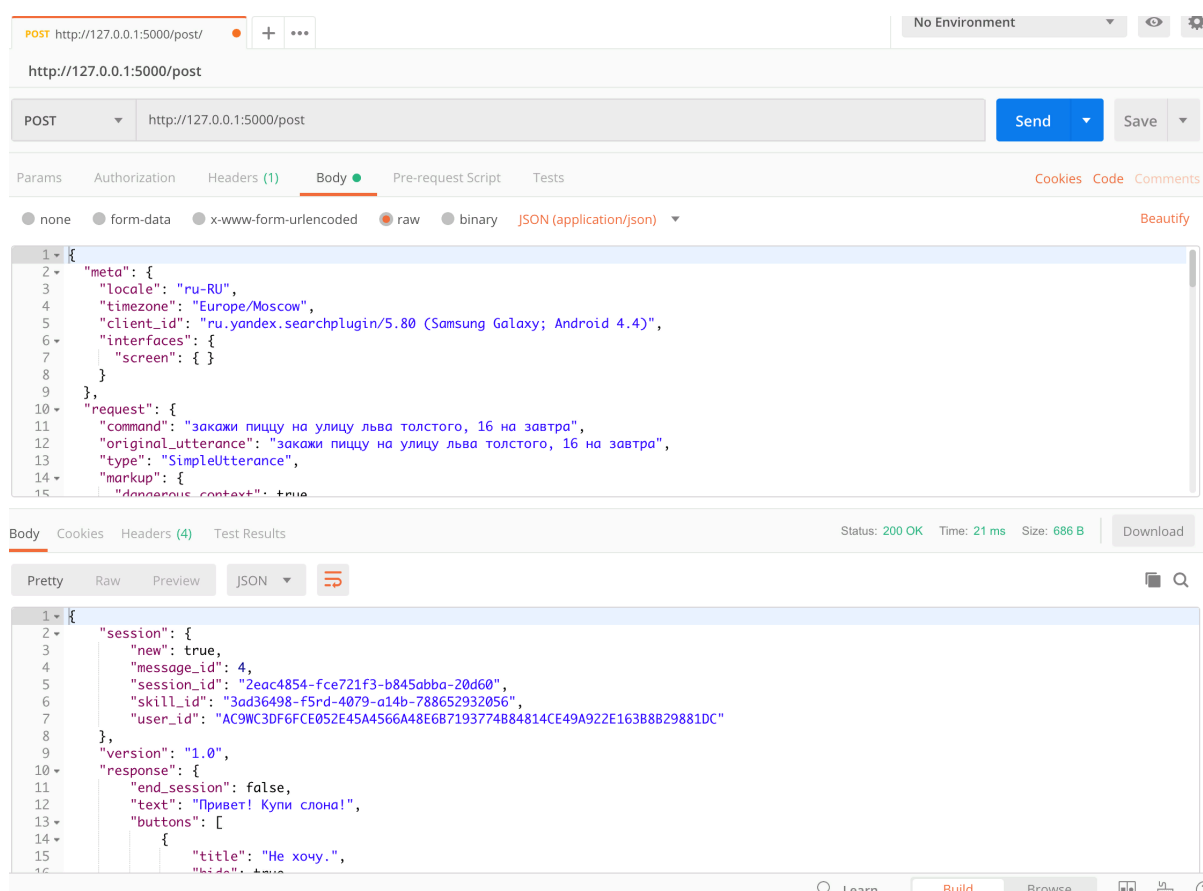
## 6. Отладка с Postman

Перед тем как продолжить, нам надо убедиться, что созданный нами WEB-сервер возвращает правильный ответ (или похожий на правильный). Воспользуемся для этого программой Postman.

**Postman** — это приложение, которое позволяет взаимодействовать с API без написания кода. Это бывает удобно, чтобы находить ошибки и «пробовать» новое API перед разработкой. Сейчас мы проверим ответ от нашего приложения.

Запустите созданное ранее приложение в среде разработки и скопируйте адрес вида `http://127.0.0.1:5000/post` в строку запроса программы Postman.

JSON запроса можно взять из документации Алисы или на крайний случай из рассмотренного нами примера.



Проверьте, что ваш ответ похож на тот, что требует документация Алисы. Если это так, то поздравляем, мы добились результата :)

## 7. Pythonanywhere — запускаем наше приложение

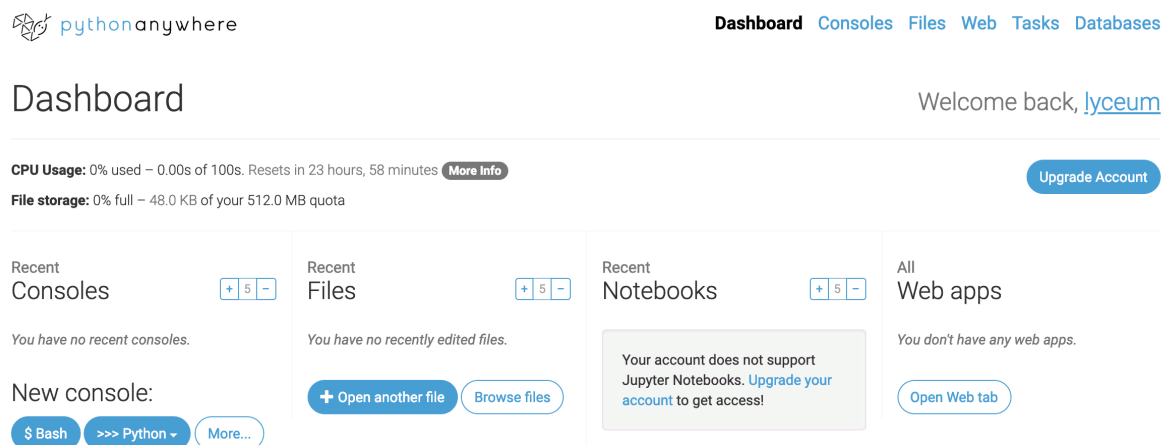
Пока наш навык «не живёт» в интернете. Мы только проверили его работу локально. Настало время научиться размещать навыки во всемирной паутине. Это можно сделать несколькими способами:

1. Разместить свой компьютер у провайдера. *Достаточно сложный способ, требующий навыков в администрировании серверов.*
2. Приобрести **виртуальный частный сервер (VPS)** — виртуальную машину с операционной системой, на которую надо установить всё ПО для работы нашего навыка.
3. Воспользоваться готовыми виртуальными машинами с уже установленным ПО.

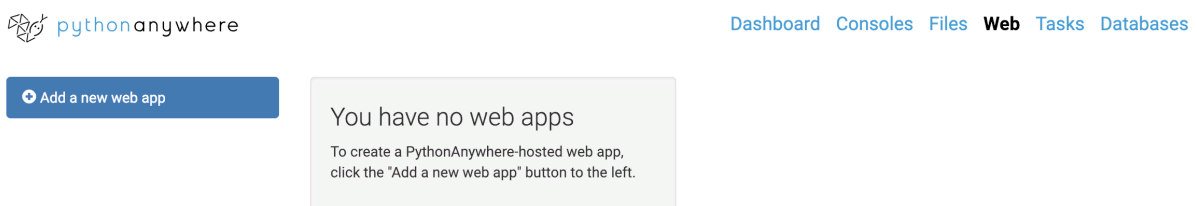
Мы пойдём по последнему пути и воспользуемся сервисом **PythonAnyWhere**. Однако следует иметь в виду, что для профессиональной работы всё же надо использовать 1 или 2 варианта, например, воспользоваться Яндекс.Облаком, которое предоставляет двухмесячный тестовый период.

Выполните следующие действия:

1. Перейдите по [ссылке](#) и нажмите **Create a Beginner account**.
2. Пройдите регистрацию, подтвердите e-mail и авторизуйтесь. В итоге вы должны увидеть вот такую панель:



3. В правом верхнем углу в меню выберите раздел **Web** и нажмите **Add a new web app**.



4. При заполнении выберите тип приложения **flask** и последнюю версию **python**. Остальные поля оставьте без изменений.



+ Add a new web app

Create new web app

## Select a Python Web framework

...or select "Manual configuration" if you want detailed control.

- » Django
- » web2py
- » Flask
- » Bottle
- » Manual configuration (including virtualenvs)

What other frameworks should we have here? Send us some feedback using the link at the top of the page!

Cancel

« Back

Next »

+ Add a new web app

Create new web app

## Select a Python version

- » Python 2.7 (Flask 1.0.2)
- » Python 3.4 (Flask 1.0.2)
- » Python 3.5 (Flask 1.0.2)
- » Python 3.6 (Flask 1.0.2)
- » Python 3.7 (Flask 1.0.2)

**Note:** If you'd like to use a different version of Flask to the default version, you can use a virtualenv for your web app. There are [instructions here](#).

Cancel

« Back

Next »

5. Теперь в верхнем правом меню выбираем раздел **Files** и *кликаем* на папку **mysite**.

/home/ lyceum

Open Bash console here

0% full – 76.0 KB of your 512.0 MB quota

Directories

Enter new directory name

New directory

.local/  
mysite/



Files

Enter new file name, eg hello.py

New file

.bashrc	2019-03-07 08:19	559 bytes
.gitconfig	2019-03-07 08:19	266 bytes
.profile	2019-03-07 08:19	79 bytes
.pythonstartup.py	2019-03-07 08:19	77 bytes
.vimrc	2019-03-07 08:19	4.6 KB
README.txt	2019-03-07 08:19	232 bytes

Upload a file

100MiB maximum size

## Directories

 [New directory](#)[\\_\\_pycache\\_\\_/](#)

## Files

 [New file](#)[flask\\_app.py](#) 2019-03-07 08:22 186 bytes

Upload a file

100MiB maximum size

## 6. Загружаем наш файл:

pythonanywhere /home/lyceum/mysite/flask\_app.py (unsaved changes)

```
1 from flask import Flask, request
2 import logging
3 import json
4
5 app = Flask(__name__)
6 logging.basicConfig(level=logging.INFO)
7
8 # Хранилище данных о сессиях.
9 sessionStorage = {}
10
11
12 @app.route('/post', methods=['POST'])
13 # Функция получает тело запроса и возвращает ответ.
14 def main():
15     logging.info('Request: %r', request.json)
16
17     response = {
18         'session': request.json['session'],
19         'version': request.json['version'],
20         'response': {
21             'end_session': False
22         }
23     }
24
```

Keyboard shortcuts: Normal Share Save Save as... >>> Run

7. Возвращаемся обратно в раздел **Web** и перезапускаем наше приложение:

lyceum.pythonanywhere.com

Add a new web app

## Configuration for lyceum.pythonanywhere.com

Reload:

Reload lyceum.pythonanywhere.com

Best before date:

We're happy to host your free website – and keep it free – for as long as you want to keep it running, but you'll need to log in at least once every three months and click the "Run until 3 months from today" button below. We'll send you an email a week before the site is disabled so that you don't forget to do that. [See here for more details.](#)

## 8. Теперь наше API доступно по адресу:

lyceum.pythonanywhere.com

Add a new web app

## Configuration for lyceum.pythonanywhere.com

Reload:

Reload lyceum.pythonanywhere.com

Best before date:

We're happy to host your free website – and keep it free – for as long as you want to keep it running, but you'll need to log in at least once every three months and click the "Run until 3 months from today" button below. We'll send you an email a week before the site is disabled so that you don't forget to do that. [See here for more details.](#)

This site will be disabled on Friday 07 June 2019

Run until 3 months from today

Paying users' sites stay up forever without any need to log in to keep them running.

Скопируйте его, он нам пригодится.

Чтобы посмотреть логи, которые пишет ваша программа на Pythonanywhere, зайдите в раздел **Web**. Открутите вниз и выберите ссылку, которая заканчивается на error.log.

#### Log files:

---

The first place to look if something goes wrong.

Access log: [lyceum.pythonanywhere.com.access.log](https://lyceum.pythonanywhere.com/access.log)

Error log: [lyceum.pythonanywhere.com.error.log](https://lyceum.pythonanywhere.com/error.log)

Server log: [lyceum.pythonanywhere.com.server.log](https://lyceum.pythonanywhere.com/server.log)

Log files are periodically rotated. You can find old logs here: [/var/log](#)

## 8. Регистрация и тестирование навыка

Перейдём к завершающему и самому простому пункту. Это регистрация навыка в Алисе.

1. Зарегистрируйтесь в Яндексe или залогиньтесь, если вы уже зарегистрированы.
2. Перейдите по ссылке [Научите Алису новому.](#)
3. Нажмите **создать диалог**.
4. Выберите вариант **Навык в Алисе**.
5. Заполните «активационное имя». По этой фразе Алиса будет вызывать ваш навык.
6. Заполните поля, указав ссылку в поле Webhook URL.

## Настройки

Черновик

Опубликованная версия

### Основные настройки

Название \*

Купи слона

Название диалога, которое будет отображаться на странице в каталоге Алисы

Активационное имя \*

Уникальное имя навыка, с помощью которого пользователь сможет вызвать диалог. Может совпадать с названием диалога, либо являться его расшифровкой. Пример полной фразы активации: «Запусти навык [Активационное имя]». ?

Webhook URL \*

<https://lyceum.pythonanywhere.com/post>

Адрес, на который будут отправляться запросы ?

Яндекс.Облако

☐ Нужен грант на Яндекс.Облако

Запросить грант на хостинг в Яндекс.Облаке на 1 год (сначала убедитесь, что ваш навык перенесён в Яндекс.Облако) ?

Голос \*

Оксана



Перейдите на вкладку **Тестирование** и проверьте работу своего навыка :)

Помощь

© 2018 – 2019 ООО «Яндекс»