

Вау! ИИ готовит к ЕГЭ по информатике

[Попробовать](#)

Новый конкурс! Нарисуйте [комикс](#) или [инфографику](#) на любую тему из области информатики. Приём работ до 17 марта.

[← Урок Алиса №2](#)

Алиса — урок 2

- 1 Введение
- 2 Картинка в ответе
- 3 Навык, который знакомится
- 4 Игра
- 5 Пользуемся API Яндекс.Карт
- 6 Получаем координаты города
- 7 Получаем страну города
- 8 Рассчитываем расстояние от города до города
- 9 Програмируем навык
- 10 Функции Яндекс.Облака

Аннотация

Сегодня мы продолжим разрабатывать навыки для Алисы: научим ее знакомиться с пользователем и взаимодействовать с внешними API. В качестве бонуса расскажем, как разместить свой навык в Яндекс.Облаке.

1. Введение

На этом уроке мы напишем игру «Угадай город». Алиса показывает пользователю фотографию города, а он должен его отгадать.

Важный момент: перед игрой пользователь должен представиться, а Алиса — его поприветствовать. Это добавит нашему навыку немного человечности.

Находить именованные сущности (город и имя) мы уже умеем, осталось научиться отправлять пользователю картинку.

Итак, приступим.

2. Картинка в ответе

Научимся показывать картинку в ответе. Например, для нашей игры нужно будет демонстрировать картинку города.

Для начала рассмотрим JSON ответа Алисы, в котором прикреплена картинка:

```
{
  "response": {
    "text": "Здравствуйте! Это мы, хорооводоведы.",
    "tts": "Здравствуйте! Это мы, хоров+одо в+еды.",
    "card": {
      "type": "BigImage",
      "image_id": "1027858/46r960da47f60207e924",
      "title": "Заголовок для изображения",
      "description": "Описание изображения.",
      "button": {
        "text": "Надпись на кнопке",
        "url": "http://example.com/",
        "payload": {}
      }
    },
    "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"
}
```

В отличие от примера из предыдущего урока, в ответе появился раздел `card`. Он и содержит в себе картинку.

Рассмотрим подробно его содержание:

```
"card": {
  "type": "BigImage",
  "image_id": "1027858/46r960da47f60207e924",
  "title": "Заголовок для изображения",
  "description": "Описание изображения.",
  "button": {
    "text": "Надпись на кнопке",
```

```
"url": "http://example.com/",  
"payload": {}  
}
```

Мы опишем только те поля, которые будем использовать. Про оставшиеся можно посмотреть в [документации Алисы](#).

`card` — описание карточки-сообщения с поддержкой изображений. Если приложению удастся отобразить карточку для пользователя, свойство `response.text` не используется

– `type` — тип карточки. Поддерживаемые значения:

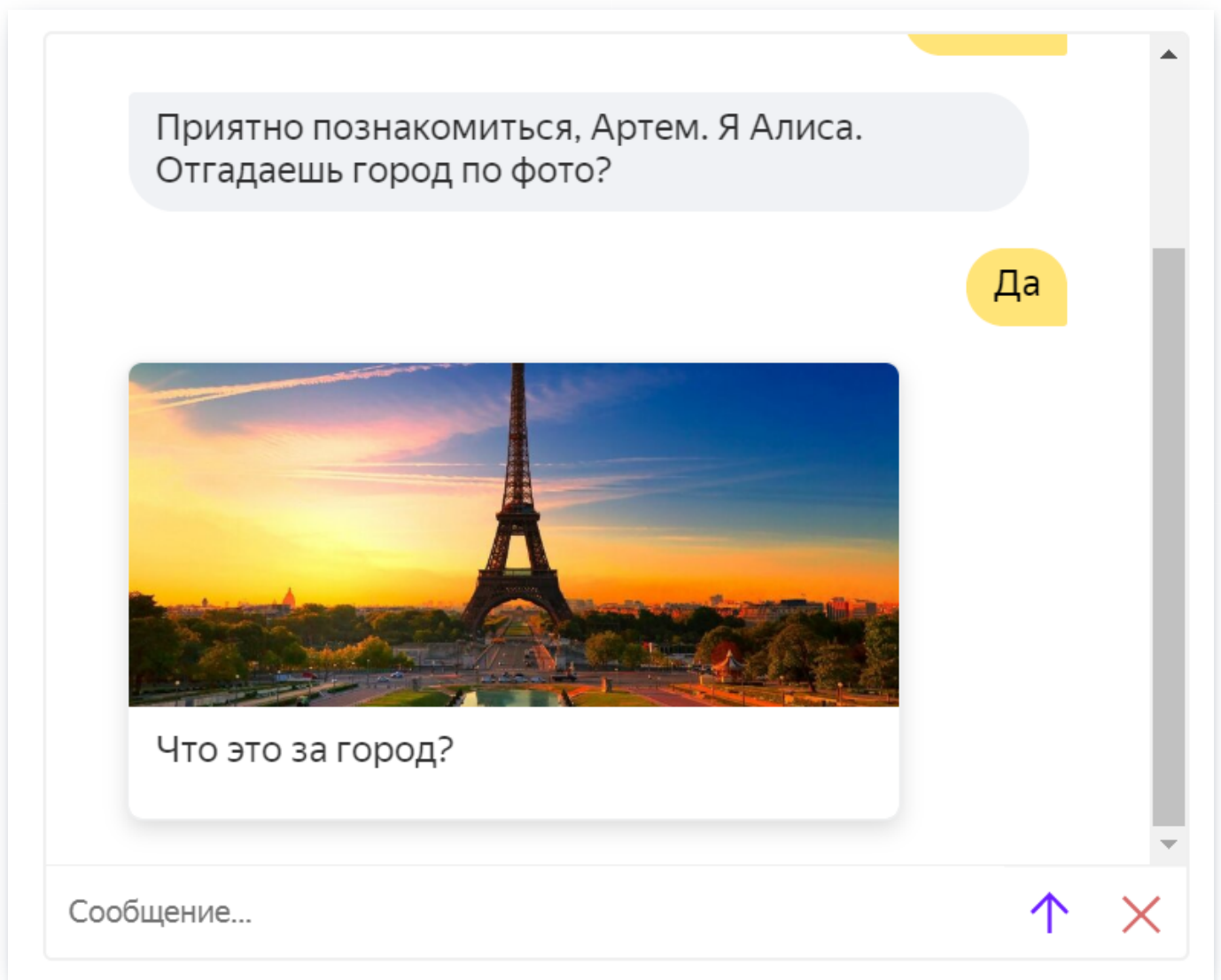
- `BigImage` — одно изображение
- `ItemsList` — галерея изображений (от 1 до 5)

Требуемый формат ответа зависит от типа карточки. Мы будем использовать `BigImage`

- `image_id` — идентификатор изображения, который возвращается в ответ на запрос загрузки
- `title` — заголовок для изображения. Это на самом деле увидит пользователь

В этом [архиве](#) мы собрали для вас картинки, которые можно использовать для решения задач урока (но можете использовать свои изображения). Подробно про то, как загружать ресурсы к своему навыку, мы говорили на прошлом уроке.

Вот как может выглядеть ответ Алисы с картинкой:



3. Навык, который знакомится

Напишем простой навык, который ведет следующий диалог:

- Привет! Назови свое имя!
- Саша.
- Приятно познакомиться, Саша. Я — Алиса. Какой город хочешь увидеть?
- Москва (Нью-йорк, Париж).
- Этот город я знаю. *И показывает фото города*

Если вдруг пользователь не назвал имя, Алиса должна сказать:

- Не расслышала имя. Повтори, пожалуйста!

Если пользователь не назвал город или назвал город, для которого нет картинки, Алиса должна сказать:

- Первый раз слышу об этом городе. Попробуй еще разок!

Код приведен ниже:

```
from flask import Flask, request, jsonify
import logging
import json
import random

app = Flask(__name__)

logging.basicConfig(level=logging.INFO)

# создаем словарь, в котором ключ — название города,
# а значение — массив, где перечислены id картинок,
# которые мы записали в прошлом пункте.

cities = {
    'москва': ['1540737/daa6e420d33102bf6947',
               '213044/7df73ae4cc715175059e'],
    'нью-йорк': ['1652229/728d5c86707054d4745f',
                 '1030494/aca7ed7acefde2606bdc'],
    'париж': ['1652229/f77136c2364eb90a3ea8',
              '3450494/aca7ed7acefde22341bdc']
}

# создаем словарь, где для каждого пользователя
# мы будем хранить его имя
sessionStorage = {}

@app.route('/post', methods=['POST'])
def main():
    logging.info(f'Request: {request.json!r}')
    response = {
        'session': request.json['session'],
        'version': request.json['version'],
```

```

        'response': {
            'end_session': False
        }
    }
}
handle_dialog(response, request.json)
logging.info(f'Response: {response!r}')
return jsonify(response)

def handle_dialog(res, req):
    user_id = req['session']['user_id']

    # если пользователь новый, то просим его представиться.
    if req['session']['new']:
        res['response']['text'] = 'Привет! Назови свое имя!'
        # создаем словарь в который в будущем положим имя пользователя
        sessionStorage[user_id] = {
            'first_name': None
        }
        return

    # если пользователь не новый, то попадаем сюда.
    # если поле имени пустое, то это говорит о том,
    # что пользователь еще не представился.
    if sessionStorage[user_id]['first_name'] is None:
        # в последнем его сообщении ищем имя.
        first_name = get_first_name(req)
        # если не нашли, то сообщаем пользователю что не расслышали.
        if first_name is None:
            res['response']['text'] = \
                'Не расслышала имя. Повтори, пожалуйста!'
        # если нашли, то приветствуем пользователя.
        # И спрашиваем какой город он хочет увидеть.
        else:
            sessionStorage[user_id]['first_name'] = first_name
            res['response']['text'] = 'Приятно познакомиться, ' \
                + first_name.title() \
                + '. Я - Алиса. Какой город хочешь увидеть?'
            # получаем варианты buttons из ключей нашего словаря cities
            res['response']['buttons'] = [
                {
                    'title': city.title(),
                    'hide': True
                } for city in cities
            ]

    # если мы знакомы с пользователем и он нам что-то написал,
    # то это говорит о том, что он уже говорит о городе,
    # что хочет увидеть.
    else:
        # ищем город в сообщении от пользователя
        city = get_city(req)
        # если этот город среди известных нам,

```

```

# то показываем его (выбираем одну из двух картинок случайно)
if city in cities:
    res['response']['card'] = {}
    res['response']['card']['type'] = 'BigImage'
    res['response']['card']['title'] = 'Этот город я знаю.'
    res['response']['card']['image_id'] = random.choice(cities[city])
    res['response']['text'] = 'Я угадал!'
# если не нашел, то отвечает пользователю
# 'Первый раз слышу об этом городе.'
else:
    res['response']['text'] = \
        'Первый раз слышу об этом городе. Попробуй еще разок!'

def get_city(req):
    # перебираем именованные сущности
    for entity in req['request']['nlu']['entities']:
        # если тип YANDEX.GEO то пытаемся получить город(city),
        # если нет, то возвращаем None
        if entity['type'] == 'YANDEX.GEO':
            # возвращаем None, если не нашли сущности с типом YANDEX.GEO
            return entity['value'].get('city', None)

def get_first_name(req):
    # перебираем сущности
    for entity in req['request']['nlu']['entities']:
        # находим сущность с типом 'YANDEX.FIO'
        if entity['type'] == 'YANDEX.FIO':
            # Если есть сущность с ключом 'first_name',
            # то возвращаем ее значение.
            # Во всех остальных случаях возвращаем None.
            return entity['value'].get('first_name', None)

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

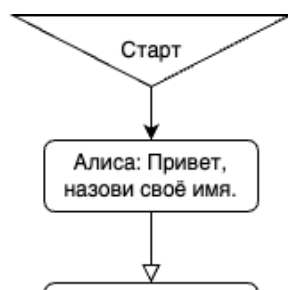
```

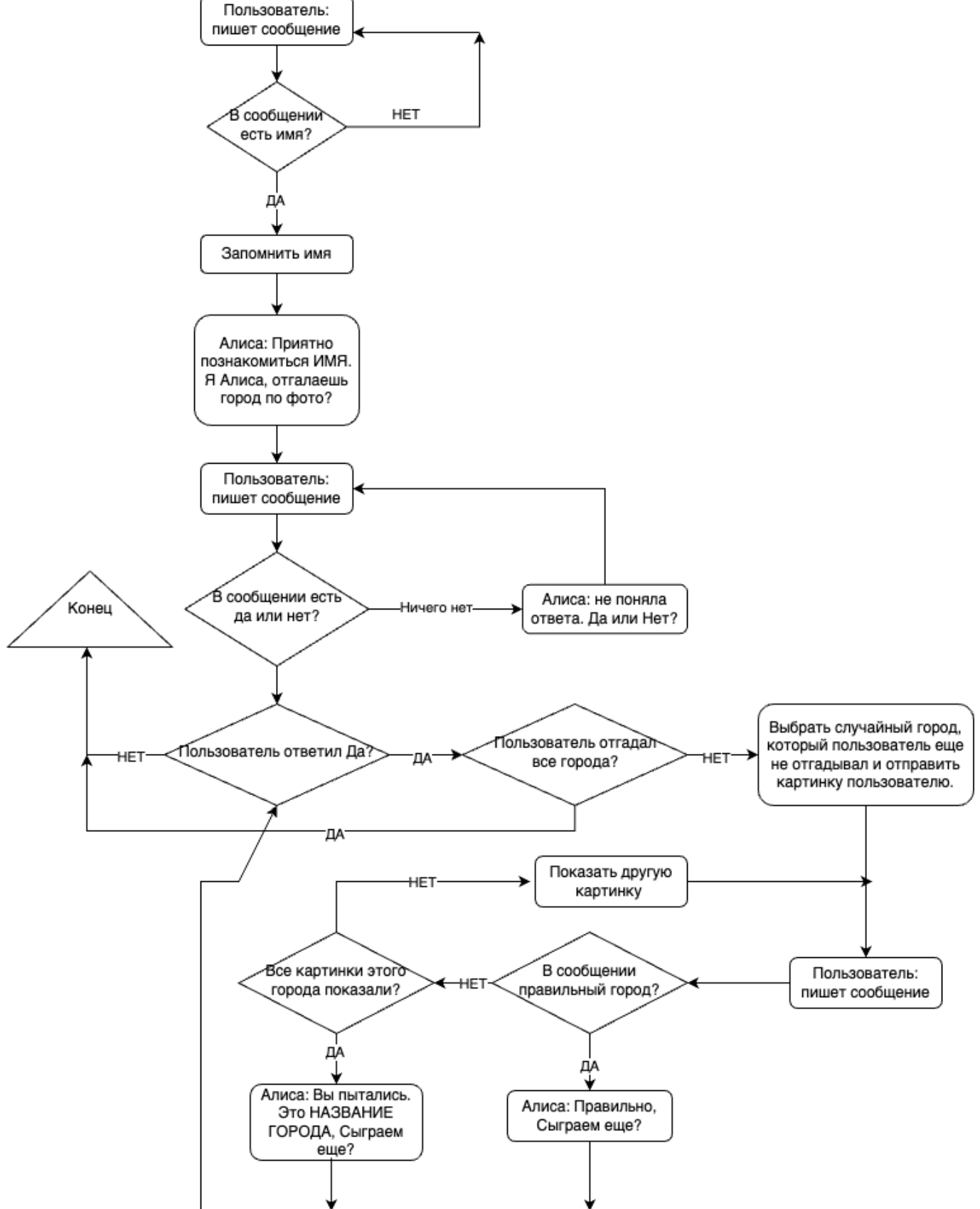
Загружаем этот код в Glitch или делаем туннель в ngrok, заполняем данные в Алисе и отправляемся тестировать!

4. Игра

Доработаем предыдущую программу до конца.

Рассмотрим блок-схему, чтобы понять логику общения нашего навыка с пользователем.





Код игры приведен в [файле](#).

5. Пользуемся API Яндекс.Карт

Сегодня мы попробуем совместить наш опыт по работе со сторонними API (Яндекс.Карты) и знания Алисы, чтобы создать новый навык, который будет сообщать пользователю, в какой стране находится загаданный город, а также вычислять расстояние от одного города до другого.

Алгоритм работы навыка следующий:

- Если мы пишем название одного города, Алиса сообщит нам, в какой стране находится этот город

- Если мы пишем названия двух городов, Алиса посчитает расстояние между ними
- Если мы вдруг напишем названия трех и более городов, Алиса возмутится и сообщит: «Слишком много городов. Я запуталась»

Для получения информации о географических объектах мы воспользуемся **геокодером Яндекс.Карт**.

6. Получаем координаты города

Мы уже знаем, что Алиса умеет вычленять из текста разные сущности, в том числе и названия городов.

Напишем функцию `get_coordinates(city_name)`, которая получает географические координаты города по его имени.

Напомним: надо отправить **HTTP-запрос**, например, такой `https://geocode-maps.yandex.ru/1.x/?geocode=Москва&format=json` Яндекс.Картам, а потом разобрать ответ.

Возвращает эта функция кортеж с координатами города. В случае, если в процессе работы произошла **любая ошибка**, мы вернем исключение.

```
import requests

def get_coordinates(city_name):
    try:
        # url, по которому доступно API Яндекс.Карт
        url = "https://geocode-maps.yandex.ru/1.x/"
        # параметры запроса
        params = {
            "apikey": "40d1649f-0493-4b70-98ba-98533de7710b",
            # город, координаты которого мы ищем
            'geocode': city_name,
            # формат ответа от сервера, в данном случае JSON
            'format': 'json'
        }
        # отправляем запрос
        response = requests.get(url, params)
        # получаем JSON ответа
        json = response.json()
        # получаем координаты города
        # (там написаны долгота(longitude), широта(latitude) через пробел)
        # посмотреть подробное описание JSON-ответа можно
        # в документации по адресу https://tech.yandex.ru/maps/geocoder/
        coordinates_str = json['response']['GeoObjectCollection'][
            'featureMember'][0]['GeoObject']['Point']['pos']
        # Превращаем string в список, так как
        # точка - это пара двух чисел - координат
        long, lat = map(float, coordinates_str.split())
        # Вернем ответ
        return long, lat
    except Exception as e:
        return e
```


7. Получаем страну города

Функция `get_country(city_name)` вернет нам страну, в которой находится указанный город. Отличие от предыдущей функции заключается лишь в получении других данных из ответа геокодера.

```
def get_country(city_name):
    try:
        url = "https://geocode-maps.yandex.ru/1.x/"
        params = {
            "apikey": "40d1649f-0493-4b70-98ba-98533de7710b",
            'geocode': city_name,
            'format': 'json'
        }
        data = requests.get(url, params).json()
        # все отличие тут, мы получаем имя страны
        return data['response']['GeoObjectCollection'][
            'featureMember'][0]['GeoObject']['metaDataProperty'][
                'GeocoderMetaData']['AddressDetails']['Country']['CountryName']
    except Exception as e:
        return e
```

8. Рассчитываем расстояние от города до города

А вот для вычисления расстояний между двумя точками необходимы знания **тригонометрических** функций. Ведь Земля — круглая!

Подробно о расчете коротких расстояний на Земле можно прочитать [тут](#). Пока же можно просто воспользоваться приведенной функцией.

```
import math

def get_distance(p1, p2):
    # p1 и p2 - это кортежи из двух элементов - координаты точек
    radius = 6373.0

    lon1 = math.radians(p1[0])
    lat1 = math.radians(p1[1])
    lon2 = math.radians(p2[0])
    lat2 = math.radians(p2[1])

    d_lon = lon2 - lon1
    d_lat = lat2 - lat1

    a = math.sin(d_lat / 2) ** 2 + math.cos(lat1) * math.cos(lat2) * math.sin(d_lon / 2) ** 2
    c = 2 * math.atan2(a ** 0.5, (1 - a) ** 0.5)

    distance = radius * c
    return distance
```

9. Программируем навык

Наша программа будет состоять из двух файлов.

В первом файле мы разместим код, который будет отвечать за общение с Алисой, а во втором — функции, которые связаны с общением с API Яндекс.Карт.

Всю функциональность общения с картами мы уже реализовали, поэтому осталось объединить все функции в один файл — **geo.py**.

А во втором файле мы расположим код, который отвечает за общение с Алисой — **app.py**.

Многое из того, что мы делаем тут, уже было сделано в других уроках. Так что объясним только новые моменты.

```
from flask import Flask, request, jsonify
import logging
import json
# импортируем функции из нашего второго файла geo
from geo import get_country, get_distance, get_coordinates

app = Flask(__name__)

# Добавляем логирование в файл.
# Чтобы найти файл, перейдите на pythonwhere в раздел files,
# он лежит в корневой папке
logging.basicConfig(level=logging.INFO, filename='app.log',
                    format='%(asctime)s %(levelname)s %(name)s %(message)s')

@app.route('/post', methods=['POST'])
def main():
    logging.info('Request: %r', request.json)
    response = {
        'session': request.json['session'],
        'version': request.json['version'],
        'response': {
            'end_session': False
        }
    }
    handle_dialog(response, request.json)
    logging.info('Request: %r', response)
    return jsonify(response)

def handle_dialog(res, req):
    user_id = req['session']['user_id']
    if req['session']['new']:
        res['response']['text'] = \
            'Привет! Я могу показать город или сказать расстояние между городами!'
        return
    # Получаем города из нашего
    cities = get_cities(req)
    if not cities:
        res['response']['text'] = 'Ты не написал название ни одного города!'
    elif len(cities) == 1:
        res['response']['text'] = 'Этот город в стране - ' + \
```

```

        get_country(cities[0])

elif len(cities) == 2:
    distance = get_distance(get_coordinates(
        cities[0]), get_coordinates(cities[1]))
    res['response']['text'] = 'Расстояние между этими городами: ' + \
        str(round(distance)) + ' км.'

else:
    res['response']['text'] = 'Слишком много городов!'

def get_cities(req):
    cities = []
    for entity in req['request']['nlu']['entities']:
        if entity['type'] == 'YANDEX.GEO':
            if 'city' in entity['value']:
                cities.append(entity['value']['city'])
    return cities

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

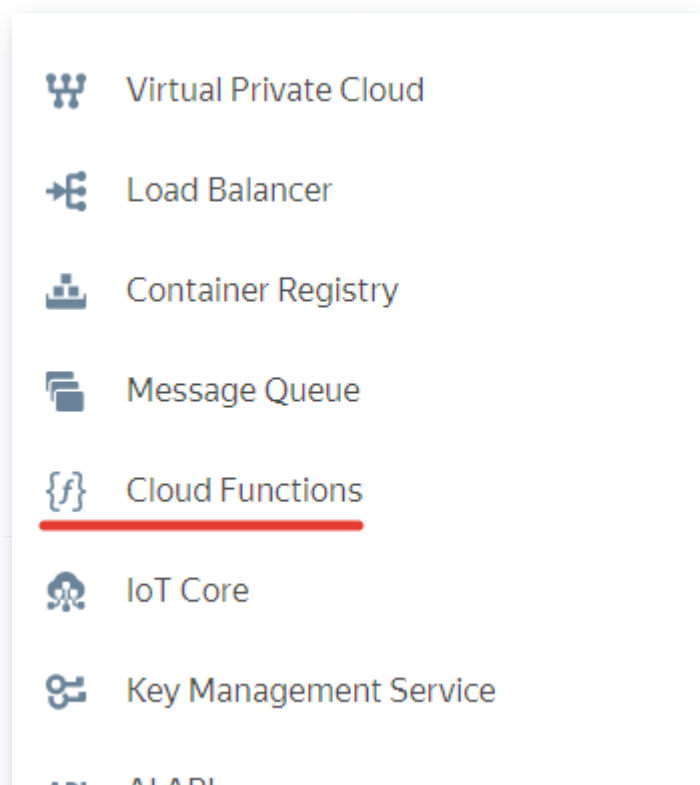
```

Осталось протестировать и задеплоить наш навык.

10. Функции Яндекс.Облака

Яндекс.Облако добавило возможность создавать свои функции в облаке без необходимости настраивать виртуальную машину и различные права доступа. У функций ограниченная функциональность, но вполне достаточная для, например, небольших навыков для голосового помощника Алиса. Большим плюсом использования функций с этой целью является то, что в этом случае они **не тарифицируются**, то есть пользователь может размещать сколько угодно таких функций бесплатно.

Давайте посмотрим, как сделать свою функцию. Зайдем в Яндекс.Облако и выберем пункт меню Cloud Functions.



API AI API

Monitoring

DataLens

Мы рассмотрим пример создания функции для самого простого навыка, который просто повторяет пользовательский ввод. Поэтому введем название и описание навыка исходя из его функциональности:

Создание функции

Имя ?

repeater-alice

Описание ?

Навык-повторюн для Алисы

Создать

Отмена

После этого создадим функцию. Файл можно создать как в редакторе, так и сделать его на своем компьютере, а потом загрузить в облако zip-архив.

Сама функция будет немного отличаться от того кода, который мы привыкли делать с использованием flask. Вот пример кода файла main.py:

```
def handler(event, context):
    """
    Точка входа для облачной функции.
    :param event: содержимое request.json().
    :param context: информация о текущем контексте выполнения.
    :return: ответ будет представлен в виде json автоматически.
    """

    text = 'Привет, я повторю все, что вы скажете'
    if 'request' in event and \
        'original_utterance' in event['request'] \
        and len(event['request']['original_utterance']) > 0:
        text = event['request']['original_utterance']
    return {
        'version': event['version'],
        'session': event['session'],
        'response': {
            'text': text,
            'end_session': 'false'
        },
    }
}
```

Обратите внимание: flask тут нам уже не нужен. Мы просто делаем функцию с определенной сигнатурой. Запрос к ней обработает Яндекс.Облако и положит все, что было, в `request.json()` в параметр `event`. Кроме того, нет необходимости самостоятельно формировать json в ответ, это произойдет автоматически.

Для того чтобы наша функция работала корректно, надо указать еще следующие значения в настройках:

- Среда выполнения — python37
- Точка входа — `main.handler` (имя файла без `py` + имя функции в этом файле, которую надо запускать)

После этого можно нажать кнопку «Создать версию».

На вкладке «Тестирование» можно отправить тестовый запрос в нашу функцию почти как в Postman, причем есть возможность указать шаблон данных в формате запроса Алисы.

Тестирование

Тег версии

Шаблон данных

Входные данные

```
{
  "meta": {
    "client_id": "ru.yandex.searchplugin/7.16 (none none; android)",
    "interfaces": {
      "account_linking": {},
      "payments": {},
      "screen": {}
    },
    "locale": "ru-RU",
    "timezone": "UTC"
  },
  "request": {
    "original_utterance": "user text",
    "command": "",
    "nlu": {
```

▶ Запустить тест

Протестируйте функцию.

Для того чтобы Алиса смогла отправить запрос к функции, ее надо сделать **публичной**. Этот переключатель находится на вкладке **Обзор**. Ссылку для Алисы копировать необязательно.

Теперь перейдем в настройки навыка Алисы и вместо Webhook выберем нужную публичную функцию в Яндекс.Облаке из списка:

Backend *

☐ Webhook URL

☒ Функция в Яндекс.Облаке

cloud-[REDACTED]/default/alice-repeater

Функции Cloud Functions, которые вы используете для навыков Алисы, не тарифицируются.

Убедимся, что все работает:



Чат

☐ Нет экрана

Привет, я повторю все, что вы скажете

Яндекс.Лицей

Яндекс.Лицей

Сообщение...  

То, что функция не использует flask, немного неудобно для локальной отладки. Но достаточно просто вынести ее в отдельный файл так, чтобы обработчик URL из flask вызывал ее, подстраивая данные до нужной сигнатуры и затем превращая ответ в json. Таким образом будет удобно и локально тестировать функцию, и затем заливать ее в Яндекс.Облако.

Исключительное право на учебную программу и все сопутствующие ей учебные материалы, доступные в рамках сервиса, принадлежат АНО ДПО «Образовательные технологии Яндекса». Воспроизведение, копирование, распространение и иное использование программы и материалов допустимо только с предварительного письменного согласия АНО ДПО «Образовательные технологии Яндекса».

[Пользовательское соглашение.](#)

© 2018 – 2024 ООО «Яндекс»