Фрагмент 1 демонстрирует использование Speech Recognition для распознавания короткой фразы, записанной в виде звукового файла.

Фрагмент 1 – Распознавание звука, записанного в аудио файле

```
# установка и подключение библиотеки распознавания речи
!pip install SpeechRecognition
import speech_recognition as sR
# установка и подключение модуля метрики качества в распознавании речи
!pip install jiwer
from jiwer import wer
# Подключение дополнительных библиотек
…
# создание объекта класса Recognizer
r = sR.Recognizer()
# считываем аудиофайл
audio = r.record('Путь к файлу', duration=duration)
# собственно распознавание с установкой используемого языка
result = r.recognize_google(audio, language='ru')

nuzna_fraza = 'вот такой текст' #ожидаем, что этот текст записан в звуковом файле

print('Оригинал: ', nuzna_fraza)
print('Результат распознавания:', result)
# вывод метрики качества
print('WER:', wer(nuzna_fraza.lower(), result.lower()))
```

Указанные фрагмент не включает код с подключением необходимых библиотек. Однако приведенный фрагмент кода с использованием пакета Speech Recognition при его лаконичности с достаточной полнотой демонстрирует распознавание короткой фразы из звукового файла. Использование готовых инструментов для передачи речевого взаимодействия в разрабатываемых приложениях от ведущих компаний (в их числе Google, MicroSoft, Sound Hound, IBM, PocketSphinx) избавляет разработчика от проектирования собственной нейронной сети: сбора и обработки датасета, создания нейросети и ее обучения и других ресурсоемких мероприятий.

ЛИТЕРАТУРА

1. SpeechRecognition 3.9.0 [Электронный ресурс]. – Режим доступа: https://pypi.org/project/SpeechRecognition/. – Дата доступа: 12.01.2023.

2. Первичный анализ речевых сигналов [Электронный ресурс]. – Режим доступа: https://alphacephei.com/ru/lecture1.pdf. – Дата доступа: 18.01.2023.

3. Мел-кепстральные коэффициенты (MFCC) и распознавание речи [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/140828/. – Дата доступа: 18.01.2023.

И.А. КОЛЕСНИКОВ, А.А. ГОЛУБ
УО МГПУ им. И.П. Шамякина (г. Мозырь, Беларусь)

ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ В JS

В настоящее время разработка web-сайтов все чаще осуществляется не одним человеком, а командой программистов, поэтому использование классических подходов

императивного программирования для написания кода современных сложных приложений является не оптимальной. В частности, возникают сложности с повторным использованием уже написанных элементов приложений, тестированием отдельных функций, неэкономным расходованием рабочего времени программистов. Большинство описанных выше проблем решается использованием объектно-ориентированного программирования.

Объектно-ориентированное программирование (ООП) – это методология программирования, основанная на представлении программы в виде совокупности объектов. В языке программирования JavaScript, который активно используется для создания web-сайтов, большинство элементов, в том числе и функции, рассматривается как объекты. Тем не менее, объектно-ориентированное программирование в JavaScript (JS) имеет свои особенности и представляет собой объектно-прототипное программирование (ОПП). Различия между классическим ООП и ОПП легко представить зная, что в ООП есть два основополагающих термина «Класс» и «Экземпляр» и когда в ООП языках программирования, таких как Java, создается экземпляр класса, то в дальнейшем нет возможности изменить его структуру, а в ОПП языках программирования, например, JavaScript, можно создать объект без использования классов, через его литерал:

```
Var foo = {name: "foo" };
```

Рассмотрим 4 основных принципа ООП в JavaScript: наследование, инкапсуляция, абстракция, полиморфизм.

Для реализации принципа наследования в JS у объектов предусмотрено специальное скрытое свойство [[Prototype]], которое либо равно null, либо содержит ссылку на другой объект, называемый прототипом.

В качестве примера рассмотрим два объекта: «animal», являющийся родительским, и «rabbit», который является наследником. В JavaScript эти объекты описываются следующим образом:

```
let animal = {eats: true};
let rabbit= {jumps: true};
rabbit__proto__ = animal;
alert( rabbit.eats ); //true
alert( rabbit.jumps ); //true
```

Стоит понимать, что свойства __proto__ и [[Prototype]] отличаются. Свойство __proto__ является устаревшим аналогом современных getPrototype и setPrototype.

Принцип инкапсуляции в JavaScript реализуется с помощью сокрытия состояния объекта от прямого доступа извне. В JS есть два типа полей объекта: public (открытые) и private (защищенные). К полям типа public возможен доступ из других объектов других классов. Такие поля формируют внешний интерфейс класса. Обращение к полям типа private возможно только внутри объекта (внутренний интерфейс).

Защищенные поля в JS обычно начинаются с префикса _.

Ниже представлен пример использования принципа инкапсуляции в JavaScript.

```
class CoffeMachine {
 _waterAmount = 0;
 constructor(value) {
 This._waterAmount = value;
 }
 set waterAmount(value) {
 if (value < 0)
 throw new Error(«Отриц. кол-во воды»);
 this._waterAmount = value;
```

```
}
}
 let coffeeMachine = new CoffeMachine(100);
 coffeeMachine.waterAmount = 10;
```

В данном примере поле _waterAmount является скрытым и обращение к нему возможно через метод-сеттер waterAmount().

Принцип абстракции является независимым от языка программирования, поэтому рассматриваться в данной работе не будет.

Полиморфизм предполагает возможность вызова одного и того же метода для разных объектов и при этом каждый из объектов этот метод реализует по-разному. Рассмотрим пример использования полиморфизма в JavaScript.

```
class Human{
 constructor(name){
 this.name = name;
}
 say(){
 return `Меня зовут ${this.name}, я художник`;
}
}
class Men extends Human{
 constructor(name){
 super(name)
}
}
class Coder extends Human{
 constructor(name){
 super(name)
}
 say(){
 return `Меня зовут ${this.name}, я программист`;
}
}
const vasia = new Men('Vasia');
const petia = new Coder('Petia');
console.log(vasia.say());
console.log(petia.say());
```

В примере выше дочерний объект класса Coder, переопределяет метод say(), вызванный из родительского объекта класса Human. В тоже время дочерний объект класса Men, вместо переопределения метода say(), наследует его у родительского класса.

Таким образом, JavaScript позволяет в полной мере реализовать все современные тенденции объектно-ориентированного программирования при создании web-приложений.

ЛИТЕРАТУРА

1. Видео курс JS базовый (ООП) [Electronic resource] // Information Technology Video Developer Network. – Mode of access: https://www.itvdn.com (свободный доступ). – Date of accers: 08.02.2023.

2. Инкапсуляция [Electronic resource] // javascript developers. – Mode of access: https://metanit.com/ (свободный доступ). – Date of accers: 08.02.2023.

3. ООП в JavaScript [Electronic resource] // javascript developers. – Mode of access: https://frontend-stuff.com/ (свободный доступ). – Date of accers: 08.02.2023.