МГТУ им. Н.Э. Баумана

Домашнее Здадание

по курсу «ПиКЯП»

Тема: Создание телеграмм бота Цитатника.

Руководитель

Нардид А. Н.

29.11.2023

Студент группы ИУ5-3Б

Ильин Константин

29.11.2023

Отчет 1-6 лабораторные работы:

Лабораторная работа №1:

Решение биквадратного уравнения

## Задача

Написать программу для решения биквадратного уравнения вида \(ax^4 + bx^2 + c = 0\), где \(a\), \(b\), и \(c\) - коэффициенты, вводимые пользователем.

## Решение

### 1. Описание функций

#### 1.1. `solve\_biquadratic\_equation(a, b, c)`

Эта функция принимает коэффициенты \(a\), \(b\) и \(c\) биквадратного уравнения и возвращает корни уравнения в виде кортежа. Используется дискриминант для определения количества и типа корней.

#### 1.2. `get\_coefficients()`

Эта функция запрашивает у пользователя ввод коэффициентов \(a\), \(b\) и \(c\) и возвращает список введенных значений. Обеспечивает обработку исключений для некорректного ввода.

#### 1.3. `main()`

Основная функция программы. Если в командной строке переданы аргументы, то они используются в качестве коэффициентов. В противном случае вызывается `get\_coefficients()`. Затем программа проверяет корректность введенных коэффициентов и вызывает `solve\_biquadratic\_equation()` для получения корней уравнения. Результат выводится на экран.

### 2. Параметры командной строки

Программа принимает коэффициенты \(a\), \(b\) и \(c\) как аргументы командной строки. Если аргументы не переданы или переданы некорректно, программа запрашивает ввод у пользователя.

### 3. Обработка ошибок

Программа обрабатывает ошибки ввода, такие как ввод нечисловых значений, и предлагает пользователю повторить ввод.

### 4. Пример использования

```

$ python script.py 1 -3 2

Действительные корни уравнения: 2.0 1.0

```

```

$ python script.py

Введите коэффициент A: 1

Введите коэффициент B: -3

Введите коэффициент C: 2

Действительные корни уравнения: 2.0 1.0

```

### 5. Заключение

Программа успешно решает биквадратное уравнение и выводит корни на экран. Обработка ошибок и валидация ввода обеспечивают корректное выполнение программы в различных сценариях использования.

Лабораторная работа №2:

# Лабораторная работа №2: Расстояние Левенштейна и Матрица Расстояний

## Задача

Написать программу для вычисления расстояния Левенштейна между двумя строками (`word1` и `word2`). Также программа должна выводить матрицу, отражающую расстояния между всеми возможными подстроками `word1` и `word2`.

## Решение

### 1. Описание функции

#### 1.1. `levenshtein\_distance(word1, word2)`

Эта функция вычисляет расстояние Левенштейна между строками `word1` и `word2`. Внутри используется двумерный массив `dp` для хранения значений расстояний между подстроками.

### 2. Вычисление расстояния Левенштейна

Программа заполняет матрицу `dp` значениями расстояний между всеми возможными подстроками `word1` и `word2` при помощи алгоритма Левенштейна.

### 3. Вывод матрицы расстояний

После вычисления расстояния Левенштейна программа выводит на экран матрицу `dp`. Каждая ячейка матрицы содержит значение расстояния между соответствующими подстроками.

### 4. Пример использования

word1 = "кот"

word2 = "скат"

distance = levenshtein\_distance(word1, word2)

print(f"Расстояние Левенштейна между '{word1}' и '{word2}': {distance}")

### 5. Пример вывода

```

Матрица расстояний:

[0, 1, 2, 3, 4]

[1, 1, 2, 2, 3]

[2, 2, 1, 2, 3]

[3, 3, 2, 1, 2]

```

Лабораторная работа №3-4:

### Задача №1: Генератор `field`

Этот генератор принимает список словарей и ключи, которые нужно извлечь из этих словарей. Если передан один ключ, возвращает соответствующее значение из словаря. Если передано несколько ключей, возвращает словарь только с непустыми значениями этих ключей.

\*\*Решение:\*\*

```python

def field(items, \*args):

assert len(args) > 0

for item in items:

if len(args) == 1:

for arg in args:

if arg in item and item[arg] is not None:

yield item[arg]

else:

new\_item = {key: item[key] for key in args if key in item and item[key] is not None}

if len(new\_item) > 0:

yield new\_item

```

### Задача №2: Генератор `gen\_random`

Этот генератор возвращает случайные числа в заданном диапазоне.

\*\*Решение:\*\*

```python

import random

def gen\_random(num\_count, begin, end):

for \_ in range(num\_count):

yield random.randint(begin, end)

```

### Задача №3: Итератор `Unique`

Этот итератор возвращает только уникальные элементы из переданной коллекции.

\*\*Решение:\*\*

```python

class Unique(object):

def \_\_init\_\_(self, items, \*\*kwargs):

self.seen = set()

self.data = iter(items)

self.ignore\_case = kwargs.get('ignore\_case', False)

def \_\_next\_\_(self):

while True:

item = next(self.data)

if isinstance(item, str) and self.ignore\_case:

item\_key = item.lower()

else:

item\_key = item

if item\_key not in self.seen:

self.seen.add(item\_key)

return item

def \_\_iter\_\_(self):

return self

```

### Задача №4: Сортировка чисел по модулю

Эта задача сортирует числа по их модулю.

\*\*Решение:\*\*

```python

data = [4, -30, 30, 100, -100, 123, 1, 0, -1, -4]

# С использованием lambda-функции

result = sorted(data, key=lambda x: abs(x), reverse=True)

print(result)

# Без использования lambda-функции

result\_without\_lambda = sorted(data, key=abs, reverse=True)

print(result\_without\_lambda)

```

### Задача №5: Декоратор `print\_result`

Этот декоратор выводит результат работы функции на экран.

\*\*Решение:\*\*

```python

def print\_result(func):

def wrapper(\*args, \*\*kwargs):

result = func(\*args, \*\*kwargs)

print(func.\_\_name\_\_)

if isinstance(result, list):

print('\n'.join(map(str, result)))

elif isinstance(result, dict):

print('\n'.join(f"{key} = {value}" for key, value in result.items()))

else:

print(result)

return result

return wrapper

@print\_result

def test\_1():

return 1

@print\_result

def test\_2():

return 'iu5'

@print\_result

def test\_3():

return {'a': 1, 'b': 2}

@print\_result

def test\_4():

return [1, 2]

test\_1()

test\_2()

test\_3()

test\_4()

```

### Задача №6: Контекстные менеджеры

Эти примеры демонстрируют использование классического контекстного менеджера и `contextlib` для измерения времени выполнения определенных задач.

\*\*Решение:\*\*

```python

import time

from contextlib import contextmanager

class cm\_timer\_1:

def \_\_enter\_\_(self):

self.start\_time = time.time()

return self

def \_\_exit\_\_(self, exc\_type, exc\_value, traceback):

print(f"time: {time.time() - self.start\_time}")

@contextmanager

def cm\_timer\_2():

start\_time = time.time()

yield

print(f"time: {time.time() - start\_time}")

from time import sleep

with cm\_timer\_1():

sleep(5.5)

with cm\_timer\_2():

sleep(5.5)

```

Лабораторная работа №5:

Конечно, вот небольшой отчет о том, как использовать кнопки в Telegram боте с помощью библиотеки `python-telegram-bot`.

### Создание кнопок в Telegram боте

1. \*\*Настройка окружения и импорт библиотеки\*\*

Вам потребуется установить `python-telegram-bot`. Импортируйте необходимые модули и классы для работы с ботом:

```python

from telegram import InlineKeyboardButton, InlineKeyboardMarkup, Update

from telegram.ext import Updater, CommandHandler, CallbackQueryHandler, CallbackContext

```

2. \*\*Инициализация бота\*\*

Создайте экземпляр `Updater` и настройте диспетчер для обработки входящих обновлений:

```python

updater = Updater("YOUR\_BOT\_TOKEN")

dispatcher = updater.dispatcher

```

3. \*\*Создание кнопок\*\*

В функции обработки команды `/start` создайте кнопки и прикрепите их к сообщению:

```python

def start(update: Update, context: CallbackContext) -> None:

keyboard = [

[

InlineKeyboardButton("Кнопка 1", callback\_data='button1'),

InlineKeyboardButton("Кнопка 2", callback\_data='button2'),

],

[InlineKeyboardButton("Кнопка 3", callback\_data='button3')],

]

reply\_markup = InlineKeyboardMarkup(keyboard)

update.message.reply\_text('Выберите опцию:', reply\_markup=reply\_markup)

```

4. \*\*Обработка нажатий на кнопки\*\*

Напишите функцию для обработки нажатий на кнопки:

```python

def button\_click(update: Update, context: CallbackContext) -> None:

query = update.callback\_query

query.answer()

query.edit\_message\_text(f"Вы нажали на кнопку: {query.data}")

```

5. \*\*Добавление обработчиков\*\*

Добавьте обработчики для команды `/start` и для нажатий на кнопки:

```python

dispatcher.add\_handler(CommandHandler("start", start))

dispatcher.add\_handler(CallbackQueryHandler(button\_click))

```

6. \*\*Запуск бота\*\*

Запустите бота и начните получать обновления:

```python

updater.start\_polling()

updater.idle()

```

Лабораторная работа №6:

### Обработка состояний бота

1. \*\*Добавление состояний в обработчики\*\*

Прежде всего, нужно добавить логику перехода между состояниями в обработчики команд и текстовых сообщений.

```python

def start(update: Update, context: CallbackContext) -> None:

update.message.reply\_text('Привет! Ожидаю текстовое сообщение.')

global current\_state

current\_state = states["TEXT\_INPUT"]

def stop(update: Update, context: CallbackContext) -> None:

update.message.reply\_text('Пока!')

global current\_state

current\_state = states["STOP"]

def text\_input(update: Update, context: CallbackContext) -> None:

text = update.message.text

update.message.reply\_text(f'Вы ввели: {text}')

```

2. \*\*Использование состояний в обработчиках\*\*

Внутри функций обработчиков можно добавить проверки текущего состояния и соответствующие действия:

```python

def start(update: Update, context: CallbackContext) -> None:

global current\_state

if current\_state == states["START"]:

update.message.reply\_text('Привет! Ожидаю текстовое сообщение.')

current\_state = states["TEXT\_INPUT"]

def stop(update: Update, context: CallbackContext) -> None:

global current\_state

if current\_state == states["TEXT\_INPUT"]:

update.message.reply\_text('Пока!')

current\_state = states["STOP"]

def text\_input(update: Update, context: CallbackContext) -> None:

global current\_state

if current\_state == states["TEXT\_INPUT"]:

text = update.message.text

update.message.reply\_text(f'Вы ввели: {text}')

```

3. \*\*Изменение состояний в обработчиках\*\*

После выполнения определенных действий в каждом состоянии, следует изменить состояние на основе ввода пользователя:

```python

def text\_input(update: Update, context: CallbackContext) -> None:

global current\_state

if current\_state == states["TEXT\_INPUT"]:

text = update.message.text

update.message.reply\_text(f'Вы ввели: {text}')

# Допустим, после получения текста переходим в состояние STOP

current\_state = states["STOP"]

```

4. \*\*Добавление логики в основную функцию бота\*\*

Наконец, основная функция бота должна использовать состояние для определения обработчиков:

```python

def main() -> None:

updater = Updater("YOUR\_BOT\_TOKEN")

dispatcher = updater.dispatcher

# Обработчики команд

dispatcher.add\_handler(CommandHandler("start", start))

dispatcher.add\_handler(CommandHandler("stop", stop))

# Обработчик текстовых сообщений

dispatcher.add\_handler(MessageHandler(Filters.text & ~Filters.command, text\_input))

updater.start\_polling()

updater.idle()

```