Методические указания к курсу

по языку программирования Python

Содержание

[Введение.](#__RefHeading___Toc513821571) 3

Установка3

Синтаксис5

[Срезы 6](#__RefHeading___Toc513821574)

[Строки](#__RefHeading___Toc513821575) 6

[Отступы](#__RefHeading___Toc513821576) 7

[Операторы и выражения](#__RefHeading___Toc513821577) 7

[Структуры данных 10](#__RefHeading___Toc513821578)

[Списки 1](#__RefHeading___Toc513821584)1

[Словари](#__RefHeading___Toc513821586) 11

[Пример использования Pyhon для написания простого калькулятора 1](#__RefHeading___Toc513821587)2

Задания к лабораторным работам13

Методические указания.

Введение

Python – это простой в освоении и мощный язык программирования. В нём предоставлены эффективные высокоуровневые структуры данных, эффективно реализован подход к ООП. Имея элегантный синтаксис и динамическую типизацию, являясь интерпретируемым, Python стал идеальным языком для написания сценариев и быстрой разработки приложений в различных областях и на большинстве платформ. Python – высокоуровневый язык программирования, его можно читать словно повествование на английском языке. О его свободе и открытости говорит то, что каждый модуль можно прочитать через текстовый редактор.

Python – портируемый язык, на котором можно писать на разных платформах.

Python – интерпретируемый язык. Отличие от компилируемых языков программирования (таких как С++) состоит в том, что код переводится в байткод (промежуточная форма), затем в машинный код и выполняется в то время, как компилируемый переводится сразу в машинный код и выполняется.

Одно из преимуществ Python – то, что он поддерживает и процедурно-ориентированное, и объектно-ориентированное программирование.

Где целесообразно использовать Python?

1. Веб-разработка.

2. Искусственный интеллект, машинное обучение, анализ данных и визуализация.

3. Автоматизация процессов.

Установка

Чтобы установить версию Python 3.8 (14.10.2019):

Вам понадобится скачать установщик с сайта <https://www.python.org/>

(см. Рис. 1).

Рисунок 1 – Скачивание Python 3

По окончанию загрузки установщика, запустите его двойным щелчком мыши, чтобы начать [установку Python](https://python-scripts.com/install-python-windows). Во время установки **ОБЯЗАТЕЛЬНО** поставьте флажок на «Add Python X.Y to PATH» в мастере настройки, чтобы не было проблем с доступом к Python с командной строки. Чтобы приступить к установке нажмите «Установить» (см. Рис. 2). После этого Python готов к запуску на вашей системе.

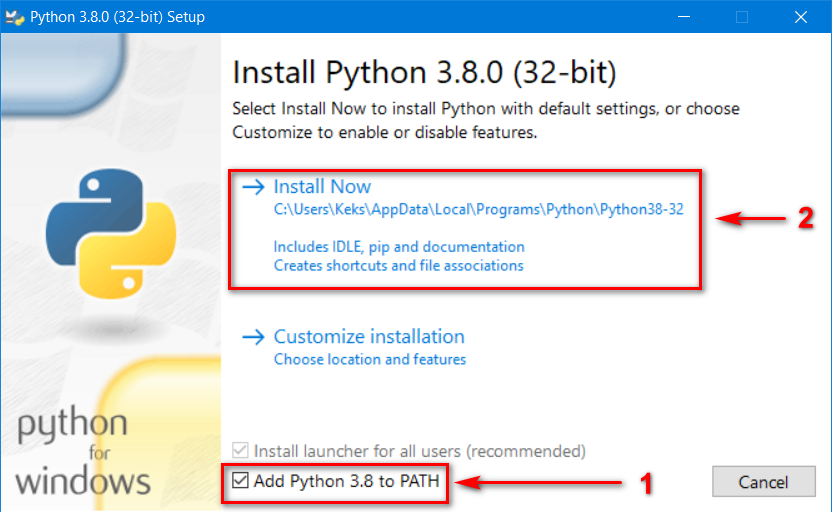
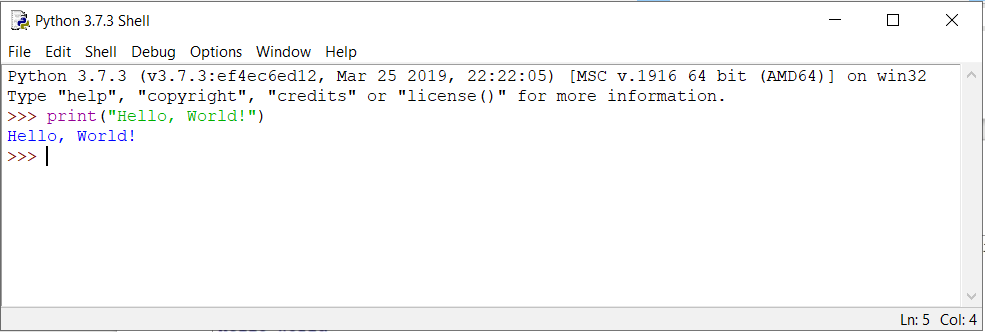


Рисунок 2 – Установка Python 3

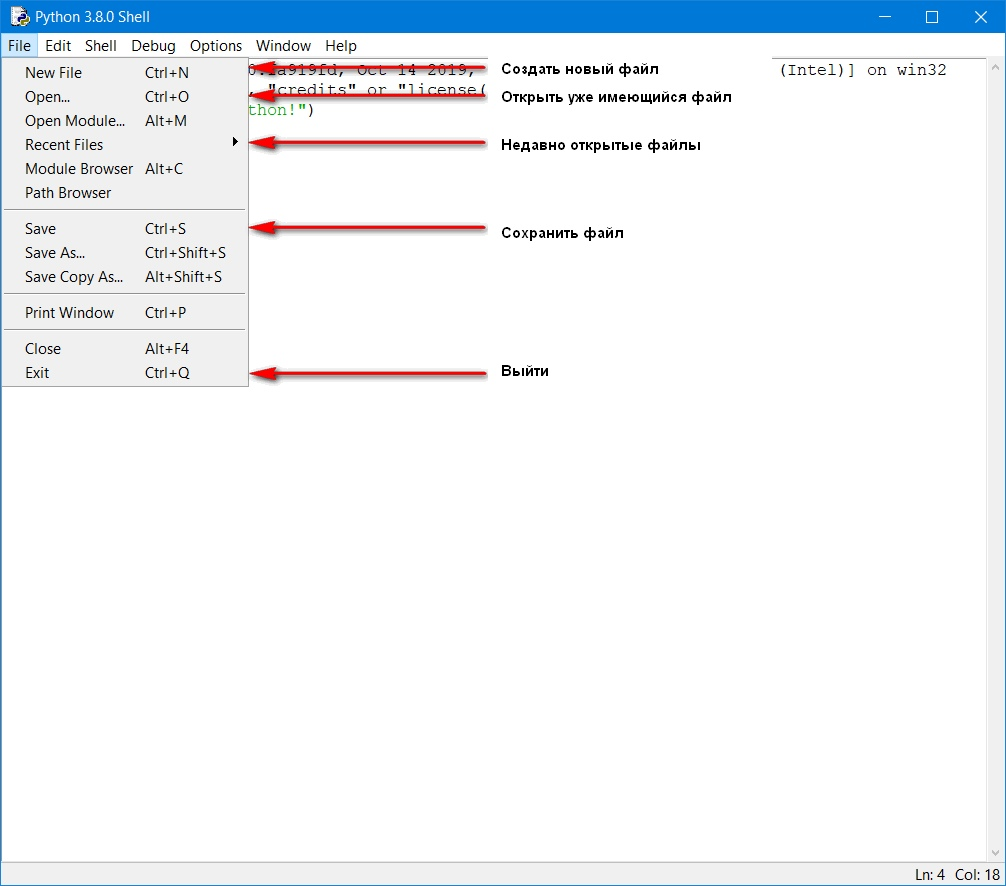
Вместе с Python устанавливается и Python IDLE, которая представляет собой интегрированную среду разработки для языка Python.

Для написания программ на Python мы **рекомендуем** использовать IDE (система программных средств, используемая программистами для разработки программного обеспечения) PyCharm (Python IDE для профессиональных разработчиков)[2]. При написании кода на Python следует соблюдать стандарты. PEP8 – это стайлгайд[3] написания кода на Python, являющийся частью языка.

Для написания первого скрипта на Python нужно всего лишь запустить IDLE (редактор кода). Код можно писать прямо в IDLE (см. Рис 3).

Рисунок 3 – Python IDLE

Для того, чтобы запустить исполняемый файл, нажмите File → Open и выберите Ваш файл с расширением .py.

Рисунок 4 – Описание меню “Файл”

Напишем в IDLE :

>>> first\_variable = 6

>>> second\_variable = 5

>>> print(first\_variable + second\_variable) # функция print - вывод в консоль результата

11 # результат

**Синтаксис**

Python – регистрозависимый язык программирования. Так, *goodname* и *goodName* – это два разных имени. Строка после символа # игнорируется интерпретатором и называется комментарием.

# это комментарий.

В Python существуют такие числовые типы: целые (например, 8, 50), с плавающей точкой (например, 3.45, 5.55) и комплексные.

Методы инициализации:

>>> a = 5 # инициализация целого числа

>>> b = 5.01 # инициализация числа с плавающей точкой

>>> c = complex(3, 4) # инициализация комплексного числа

>>> print(a, b, c)

5 5.01 (3+4j) # результат

Строки

Последовательности символов в Python называются строками. Строка должна быть заключена в кавычки. Это могут быть либо двойные, либо одинарные. Строка ‘My name is Nick’ совершенно не отличается от строки “My name is Nick”. Для того, чтобы вставить в строку кавычки, нужно обернуть саму строку другими кавычками(‘Это“пример”.’). Тройные кавычки (‘‘‘ или “““) позволяют указывать «многострочные» строки.

Чтобы составить строку на основе каких-либо данных можно использовать такую форму записи: ‘Сегодня ’ + str(date) + ‘ число’, либо метод format(). В этом методе значение каждого аргумента помещается в обозначенное место.

>>> name = "Джо"

>>> age = 29

>>> print("Возраст {} - {} лет.".format(name, age)) # вызываем метод работы со строкой format для передачи значений переменных в строку

Возраст Джо - 29 лет.

>>> print(f"Возраст {name} - {age} лет.") # в Python 3.6+ такая же функ- ция выглядит следующим образом

Возраст Джо - 29 лет.

В Python существует множество методов для работы со строками. Отдельно хотелось бы выделить:

метод find() – нахождение заданной подстроки в исходной строке:

>>> word.find('mp') # возвращает индекс первого найденного элемента или -1

3

Метод str() преобразует заданную переменную в строку, если это возможно.

Метод int() преобразует заданную переменную в целочисленное число, если это возможно.

Метод float() преобразует заданную переменную в число с плавающей точкой, если это возможно.

Срезы

Срезы – это механизм, который используется для управления строкой, списком, а именно для вырезки, обрезки.

Синтаксис:

word[новое\_начало : новый\_конец : шаг\_деления]

Отсутствие первого параметра означает начало, второго – конец, третьего – без шагов деления.

Например:

>>> word = 'example'

>>> word[3] # вывели четвертый символ строки

'm'

>>> word[3:5]

'mp'

>>> word[:5]

'examp'

>>> word[::-1] # реверс строки

'elpmaxe'

>>> word[1:5:2]

'xm'

Важность отступов в Python

Вложенные инструкции объединяются в блоки по величине отступов. Их можно записать в соответствии со следующим шаблоном:

>>> Основная инструкция:

Вложенный блок инструкций

Например:

>>> if True:

print("Условие всегда 'ИСТИНА'")

Условие всегда 'ИСТИНА'

Операторы и выражения

Как вам известно, выражение можно разделить на операнды и операторы. Операторы производят действия над операндами, т. е. данными.

Все операторы, существующие в Python, представлены в Таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Оператор | Название | Суть | Использование |
| Математические операторы | | | |
| + | Сложение | Сумма двух объектов | 4 + 2 даст 6; ‘d’ + ‘a’ даст ‘da’ |
| - | Вычитание | Разность двух чисел. Отсутствие правого операнда принимается за ноль | 8 - 7 даст 1;  -6.1 даст -6.1 |
| \* | Умножение | Произведение двух чисел. В случае со строками вернет строку, повторенную заданное число раз | 4 \* 2 даст 8;  ‘na’ \* 3 даст ‘nanana’ |
| \*\* | Возведение в степень | Вернет число, возведенное в указанную степень | 2 \*\* 4 даст 16 (так как 2 \* 2 \* 2 \* 2) |
| / | Деление | Частное двух чисел | 5 / 3 даст 1,666666667 |
| // | Целочисленное деление | Неполное частное двух чисел | 5 // 3 даст 1 |
| % | Деление по модулю | Остаток от деления | 5 % 3 даст 2 |
| Операторы сравнения являются булевыми (возвращают True или False) | | | |
| < | Меньше | Определяет истинность выражения: является ли левый операнд меньше правого | 5 < 6 даст True, следовательно условие выполнится |
| > | Больше | Является ли левый операнд больше правого | 5 > 6 даст False, следовательно условие не выполнится |
| <= | Меньше или равно | Является ли левый операнд меньше или равен правому | 5 <= 5 даст True |
| >= | Больше или равно | Является ли левый операнд больше или равен правому | 7 >= 6 даст True |
| == | Равно | Являются ли объекты одинаковыми | 8 == 9 даст False |
| != | Не равно | Являются ли объекты различными | 5 != 6 даст True |
| if | Условный оператор | Проверяет заданное условие на истинность.  Else и elif являются дополнительными условиями и не всегда могут быть выполнены | if 5 != 5:  print("5 != 5") elif 4 != 4:  print("4 != 4")  else:  print("Все условия лживы") |
| Логические операторы | | | |
| not | НЕ | Меняет логическое значение операнда на противоположное  not False даст True | k = 5  if not k:  print("yes")  else:  print(“no”)  # Выведет no, так как # выражение k == 0 # вернет False |
| and | И | Выражение будет правдивым, если оба операнда верны | if 5 == 5 and 6 == 6:  print("Верно") |
| or | ИЛИ | Выражение будет верным, если любой из операндов истинный | if 5 == 7 or 6 == 6:  print("Верно")  # 6 равно 6, поэтому # условие выполнится |
| Циклы | | | |
| while |  | Многократное выполнение блока команд, пока выполняется заданное условие. Дополнительный оператор else срабатывает, если цикл завершает свою работу. | i = 0  while i < 20:  print(str(i))  i += 1  else:  print("Конец") |
| for |  | Цикл осуществляет итерацию по последовательности объектов | ls = [‘H’, ‘E’, ‘L’, ‘L’, ‘O’]  for element in ls:  print(element) |
| break |  | Прекращение работы цикла.  При срабатывании break условие else, находящегося после цикла, не выполняется | for element in range(1, 5):  print(element)  if element==3:  break  # Выведет „1 \n 2 \n 3“ |
| continue |  | Мгновенный переход к следующей итерации | for element in range(1, 5):  if element==3:  continue  print(element)  # Выведет „1 \n 2 \n 4 # \n 5“ |

**Структуры данных**

Списки

Список – это структура данных, которая содержит упорядоченный набор элементов, т. е. хранит последовательность элементов. Примеры инициализации списка:

1)

>>> list\_ = [1, 2, 3, 4, 5]

>>> print(list\_)

[1, 2, 3, 4, 5]

2)

>>> h = "hello"

>>> list\_ = list(h)

>>> print(list\_)

['h', 'e', 'l', 'l', 'o']

3) С помощью генератора:

>>> list\_ = [x for x in range(1, 5)]

>>> print(list\_)

[1, 2, 3, 4]

Список в Python – это итерируемая структура данных, т. е. По нему можно пройтись с помощью цикла for.

>>> list\_ = [x for x in range(1, 20) if x % 2 == 0] # генерируем список с условием того, что все числа будут четными

>>> print(list\_)

[2, 4, 6, 8, 10, 12, 14, 16, 18]

>>> for i in list\_:

print(i)

2

4

6

8

10

12

14

16

18

Списки не имеют привязки к определенному типу данных, в них может храниться что угодно.

>>> list\_ = [1, "smth", [1.4, 3, 5], {'key': 'value'}]

>>> print(list\_)

[1, 'smth', [1.4, 3, 5], {'key': 'value'}]

Списки имеют большой набор функций:

* append — добавление:

>>> list\_= [1, 2]

>>> print(list\_)

[1, 2]

>>> list\_.append(3)

>>> print(list\_)

[1, 2, 3]

* index — найти индекс первого вхождения конкретного элемента:

>>> list\_ = [1, 2, 3]

>>> list\_.index(2)

1

* remove — удаление элемента:

>>> list\_.remove(1)

>>> print(list\_)

[2, 3]

* sort — сортировка:

>>> for i in range(10, -3, 2): # с помощью цикла, и функции range добавляем в список элементы упорядоченные в порядке уменьшения

list\_.append(i)

>>> print(list\_)

[2, 3, 10, 8, 6, 4, 2, 0, -2]

>>> list\_.sort()

>>> print(list\_)

[-2, 0, 2, 2, 3, 4, 6, 8, 10]

* pop — извлечение элемента, и вывод его:

>>> print(list\_)

[-2, 0, 2, 2, 3, 4, 6, 8, 10]

>>> list\_.pop(6)

6

* len — длина списка:

>>> len(list\_)

8

Существуют неизменяемые списки. Они называются кортежами. Среди преимуществ кортежей над списками можно выделить то, что первые работают быстрее.

tp = (1, 2, 3) # (1, 2, 3)

Словари

Словари – это ассоциативный массив, или хеш, в котором каждому ключу соответствует определенное значение. [ссылка на IBM] Например, реализация списка контактов в телефоне с помощью словаря: контакту с именем «Андрей» соответсвует номер «0344342323». В коде это будет выглядеть так:

contacts = {'Andrew': '0344342323'} # {'Andrew': '0344342323'}

Функции словарей:

fromkeys() — создание словаря:

{}.fromkeys(['name', 'age']) # {'age': None, 'name': None}

get() — получить значение по ключу:

print {}.get('name') # None

has\_key() — проверка существует ли значение по ключу:

{}.has\_key('name') # False

items() — возвращает список значений:

for key, value in d.items():

print(key, value)

keys() — возвращает список ключей:

d = {'url': 'no', 'title': 'good'}

d.keys() # ['url', 'title']

pop() — извлекает значение по ключу:

d.pop('title') # d {'url': 'no'}

update() — изменяет словарь:

d.update({'www':'python.org'}) # d {'www': 'python.org', 'url': 'no', 'title': 'good'}

values() — возвращает список значений:

d.values() # ['python.org', 'no', 'good']

Работа с файлами

Для открытия файла воспользуйтесь функцией open(“file\_name.format”, “mode”):

file\_name.format – имя файла, mode – режим открытия файла(r – чтение файла, w – запись в файл)

f = open(“test.txt”, “r”);

Любой открытый файл нужно закрывать функцией close().

Для полноценной работы с файлами необходимо проверять целостность файла(существует или нет, повреждён или нет), для чего лучше использовать менеджер контекста with … as. Например:

with open(“test.txt”, “r”) as f:

# ваш код

Данная конструкция гарантирует закрытие файла в любом случае.

Функции для работы с файлами:

в режиме чтения:

f.read() # считывает весь файл;

f.readline() # считывает до символа \n.

Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) — парадигма программирования, в которой главными концепциями являются понятия классов и объектов.

Класс – это тип, который описывает устройство объектов. Объект – экземпляр класса. Класс – это некий чертеж, на основании которого создаются объекты.

Объекты в ООП можно сравнить с объектами реального мира: у них также есть параметры(поля) и действия. Например:

класс: кот;

параметры: цвет, вес, рост, длина, кличка;

методы: мяукать, ходить, кушать;

объект кот1:

|  |  |
| --- | --- |
| Поля | Методы |
| цвет: рыжий | Мяукать: print(“Мяу”) |
| вес: 5 кг | Ходить: print(“Топ-топ”) |
| рост: 20 см | Кушать: print(“Ням-ням”) |
| длина: 50 см |  |
| кличка: “Рыжик” |  |

Мы создали объект кот1, который имеет свои индивидуальные характеристики. Реализуем данный пример в коде:

class Cat: # ключевое слово class - создание класса

# задание полей класса

color = str() # цвет - строка

weight = float() # вес - число с вещ. точкой

growth = float() # рост - число с вещ. точкой

length = float() # длина - число с вещ. точкой

name = str() # кличка - строка

# методы класса

def MyNameIs(self): # self - аргумент, который ссылается на объект ДАННОГО класса

print(f'Привет, меня зовут - {self.name}, мои параметры:\nРост - {str(self.growth)} см;\n'

f'Вес - {self.weight};\n Длина - {str(self.length)};\n Цвет - {self.color};\nВсем мяу (─‿‿─)')

def meow(self): # метод, выводящий на экран "Мяу"

print('Мяу')

def go(self): # метод, выводящий на экран "Топ-топ"

print('Топ-Топ')

def yum(self): # метод, выводящий на экран "Ням-ням"

print('Ням-ням')

if \_\_name\_\_ == '\_\_main\_\_':

cat1 = Cat() # создали объект первого кота,

cat1.name = 'Рыжик' # присваиваем ему кличку

cat1.length = 50.1 # длину

cat1.growth = 20.1 # рост

cat1.color = 'Рыжий' # цвет

cat1.weight = 5 # вес

cat1.MyNameIs() # вызывает метод объекта, то есть функции, которые прописаны выше в классе

cat1.meow() # ...==/==...

cat1.go() # ...==/==...

cat1.yum() # ...==/==...

print('/===================================/')

cat2 = Cat()

cat2.name = 'Мохнатик' # присваиваем ему кличку

cat2.length = 22.4 # длину

cat2.growth = 23.1 # рост

cat2.color = 'Серый' # цвет

cat2.weight = 6.1 # вес

cat2.MyNameIs() # вызывает метод объекта, то есть функции, которые прописаны выше в классе

cat2.meow() # ...==/==...

cat2.go() # ...==/==...

cat2.yum() # ...==/==...

Одним из преимуществ Python перед другими языками программирования также является наличие большого количества модулей, которые пополняются его сообществом.

Пример \* использования Python для написания простого калькулятора:

def calc(f, s, op): # создаем функцию, передаем три параметра

if op == '+': # проверяем оператор

return f + s # возвращаем значение в зависимости от операции

elif op == '-':

return f - s

elif op == '\*':

return f \* s

elif op == '/':

return f / s

else:

return 'Операция не известна'

if \_\_name\_\_ == '\_\_main\_\_':

while True: # бесконечный цикл

str\_ = input('Введите выражение в формате 5+5:> ')

tempvar = max(str\_.find(x) for x in ('+', '-', '\*', '/')) # находим индекс

# вхождения оператора

if tempvar < 0 or str\_.find(' ') != -1:

continue

try:

firstvalue = int(str\_[:tempvar])

operand = str\_[tempvar]

str\_ = str\_[tempvar + 1:]

secondvalue = int(str\_)

except: # если введена глупость

continue # следующая итерация цикла

print(calc(firstvalue, secondvalue, operand)) # вызов функции calc

**Задания к лабораторным работам**

**Лабораторная работа №1**

**Задачи на структуры данных**

Цель.

Научиться работать со структурами данных, такими как словари, списки и т. д.

Задания.

Дополнительные требования: обрабатывать вызванные пользователем исключения в программе (например, проверка на корректность ввода, проверка наличия подключаемого файла и т.д.).

*Вариант 1*.

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2. Создать список, каждый элемент которого равен кубу своего номера.

3. Определить последнее слово-палиндром в строке и вывести его.

4. Создать файл, заполнить его случайными целыми значениями. Отсортировать файл по убыванию.

5. Имеется класс student с полями ФИО и Успеваемость. Созать список этих студентов и отсортировать по успеваемости или ФИО.

*Вариант 2.*

1. Рассчитать сумму цифр степени числа 2999. Пример: 25 = 32; 3+2 = 5.

2. Определить каких чисел в списке больше: тех, которые делятся на первый элемент списка, или тех, которые делятся на последний элемент списка.

3. Вырезать из строки все цитаты.

4. Переписать в текстовый файл 1 содержимое текстового файла 2 без пустых строк, а остальные дополнить справа пробелами.

5. Создать класс с двумя переменными. Добавить метод ввода, вывода на экран этих переменных, их суммы, произведения.

*Вариант 3.*

1. Сколько вторников выпадает на второе число месяца в двадцать первом веке?

2. Найти сумму четных элементов списка, которые стоят на четных позициях.

3. Найти в строке все однокоренные слова.

4. В файле содержатся фамилии студентов и их оценки. Изменить на прописные буквы фамилии тех студентов, у которых оценка выше 3.

5. Создать класс Library. Сделать методы добавления книги, удаления книги, просмотр всех книг, сортировку по названию.

*Вариант 4.*

1. Сколько существует круговых простых чисел меньше миллиона? Круговое число – это число, перестановки цифр которого из конца в начало являются простыми числами: 197, 719, 971.

2. Найти сумму нечетных элементов списка, которые стоят на нечетных позициях.

3. Удалить в строке все “qwe”, за которыми следует четное число.

4. Удалить из текстового файла все слова, содержащие от 2 до 4 символов.

5. Создать класс «Зоомагазин». Сделать методы добавления/удаления животного, просмотр всех животных, добавить поле стоимости, методы продажи животного.

*Вариант 5.*

1. Имеется четырехзначное число. Необходимо проверить, чтобы цифры располагались по убыванию. Например: 7654 (но не 7665).

2. Найти сумму чисел списка, которые расположены до первого четного числа списка.

3. Отсортировать все слова в строке по алфавиту.

4. Создать таблицу и записать ее в файл.

5. Создать класс Аквариум, добавить методы добавления/удаления рыбок, метод замены воды. Добавить поле «время», которое постепенно будет увеличиваться, и по заданному периоду которого вода будет меняться.

*Вариант 6.*

1. Пользователь вводит четырехзначное число. Переставить цифры местами так, чтобы в конце оказались цифры меньше трёх.

2. Найти сумму чисел списка, которые расположены до первого нечетного числа списка.

3. Перевести в строке в верхний регистр все буквы “t”, с которой начинаются слова.

4. В текстовом файле изменить первую букву каждого слова на заглавную.

5. Создать класс Квадрат. Добавить методы ввода параметров квадрата, вывода квадрата в консоль (нарисовать).

*Вариант 7.*

1. Дано пятизначное число. Поменять наибольшую и наименьшую цифры местами.

2. Найти сумму чисел списка, стоящих на нечетных местах и превосходящих сумму крайних элементов списка.

3. Отсортировать строку по длине слов в ней.

4. В файле содержится текстовая строка. Определить частоту повторяемости каждой буквы в тексте и вывести её.

5. Создать класс Животное. Реализовать методы: создание, редактирование, метод «Гладить» (который будет выводить голос животного) и удаление животного.

*Вариант 8.*

1. Даны коэффициенты квадратного уравнения. Найти решение.

2. Найти сумму наибольшего и наименьшего элементов списка.

3. Даны две строки. Определить, можно ли из всех символов первой строки составить вторую строку.

4. Вывести из файла все слова, не содержащие буквы английского алфавита.

5. Создать класс Треугольник. Реализовать методы ввода сторон, проверки на треугольник, вывода типа треугольника (равнобедренный, прямоугольный и т.д.).

*Вариант 9.*

1. Пользователь вводит длины сторон треугольника. Найти его площадь и проверить существование данного треугольника.

2. Среди элементов с четными номерами найти наименьший элемент списка, кратный 3.

3. Удалить в строке все четные цифры, кратные 3.

4. Записать в текстовый файл имя абонента, его номер и город проживания. По окончании записи, вывести абонентов с одинаковыми городами на экран.

5. Создать класс Машина. Реализовать методы расчета максимальной скорости, разгона от 0 до 100 км/ч.

*Вариант 10.*

1. Даны координаты трёх вершин прямоугольника, стороны которого параллельны координатным осям. Найти координаты четвертой вершины.

2. Среди элементов с нечетными номерами найти наибольший элемент списка, кратный 5.

3. Написать функцию генерации email.

4. Переписать содержимое файла транслитом (пример: привет – privet).

5. Создать класс Торт. Задать поля ингридиетов (не менее 5), реализовать методы добавления и удаления ингридиента, расчета массы.

*Вариант 11.*

1.

2.

3.

4.

5.

*Вариант 12.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 13.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 14.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 15.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 16.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 17.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 18.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 19.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 20.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 21.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 22.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 23.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 24.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

*Вариант 25.*

1. Найти самое большое число-палиндром (читается одинаково с обеих сторон, например, 1221), полученное произведением трёх двузначных чисел.

2.

3.

4.

5.

**Лабораторная работа №2**

**Введение в Kivy**

Цель.

Получить базовые знания по использованию фреймворка Kivy.

Kivy – это библиотека[6] для быстрой разработки межплатформенных пользовательских интерфейсов. Приложения могут быть разработаны на для Linux, Windows, OS X, iOS, Android c использованием той же кодовой базы.

Установка:

Для установки Kivy на Windows вам потребуется:

Перейти в командную строку;

python -m pip install --upgrade pip wheel setuptools virtualenv

если не сработает, запустить терминал/консоль от имени администратора или прописать команды отдельно :

pip install virtualenv

pip install setuptools

pip install wheel

python -m pip install docutils pygments pypiwin32 kivy\_deps.sdl2==0.1.22 kivy\_deps.glew==0.1.12

python -m pip install kivy\_deps.gstreamer==0.1.17

python -m pip install kivy\_deps.angle==0.1.9

python -m pip install kivy==1.11.1

Если возникли какие-либо проблемы при установке, перейдите на официальный сайт Kivy[6]:

Фреймворк Kivy реализуется посредством объектно-ориентированного программирования (ООП).

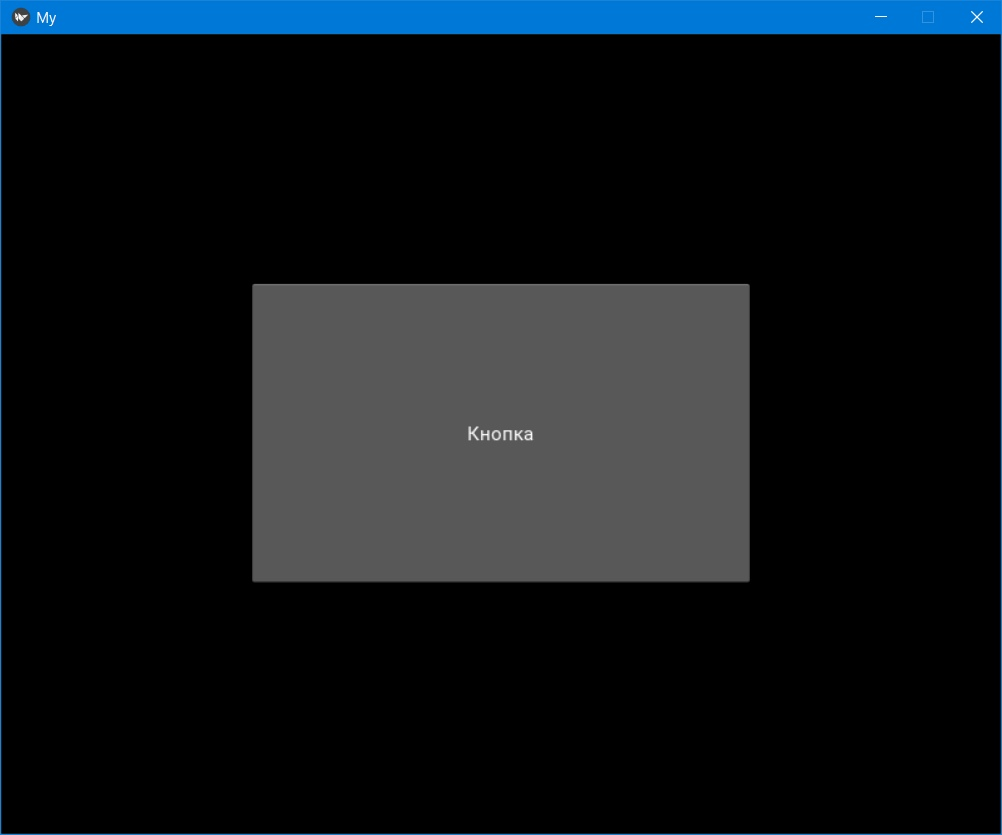
Большинство приложений Kivy начинаются с этой структуры:

from kivy.app import App  
  
class MyApp(App): # создаем класс главного окна  
 def build(self): # жизненный цикл данного окна  
 return  
MyApp().run() # запуск окна

Виджеты – это элементы, которые вы прописываете для окна, например: кнопки, поля ввода, и прочее. Подробнее о каждом из них можно прочитать на официальном сайте kivy[1].

Ниже приведён пример вывода кнопки:

def build(self): # в этой функции идет инициализация окна  
 bl = BoxLayout(orientation='vertical', # создаем Макет для виджетов(для элементов, кнопок и т.д.), задаем его ориентацию,  
 # то есть как элементы, добавленные в него будут распологаться, vertical - новый виджет ниже,  
 # horizontal - вправо  
 padding=[200, 200, 200, 200]) # отступы от краев окна, без отступов кнопка будет занимать всё окно  
 bl.add\_widget(Button(text='Кнопка')) # добавляем Кнопку, в макет, задаем кнопке текст "Кнопка"  
 return bl # возвращаем макет, ОБЯЗАТЕЛЬНО

Рисунок 3.1 – Вывод кнопки

Пример написания простейшего калькулятора с помощью Kivy:

from kivy.config import Config # для конфигурации окна  
Config.set('graphics', 'resizable', 0) # делаемым окно не изменяемым для пользователя  
Config.set('graphics', 'height', 400) # задаем высоту в 400 пикселей  
Config.set('graphics', 'width', 300) # задаем ширину в 300 пикселей  
from kivy.uix.boxlayout import BoxLayout # размещение по порядку по квадратам  
from kivy.uix.gridlayout import GridLayout # табличное размещение  
from kivy.uix.button import Button # кнопка  
from kivy.uix.textinput import TextInput # поле для ввода  
import os # работа с файлаами/ папками  
from kivy.app import App # окно  
  
  
class MyApp(App):  
 def build(self): # в этой функции идет инициализация окна  
 self.title = 'Калькулятор' # переименовываем окно  
 boxlayout = BoxLayout(orientation='vertical')  
 textinput = TextInput(hint\_text='0', # создаем поле ввода, hint\_text это текст, который показывается, когда поле для ввода пусто  
 size\_hint=(1, .25), # размеры поля ввода, по ширине полностью на всё окно, по высоте 1/4 допустимого значения от boxlayout  
 input\_filter='int', # делаем, чтоб пользователь мог ввести ТОЛЬКО целые числа  
 halign='right', # выравнивание по правой стороне  
 font\_size=40) # размер текста  
  
 gridlayout = GridLayout(cols=4) # создаем табличный макет с кол-вом колонок 4  
  
 def work\_with\_textinput(instance): # функция, которая будет вызываться по нажатию кнопок  
 if instance.text == 'С': # если нажата кнопка С, то стереть  
 textinput.text = '' # убираем весь текст из поля ввода  
 elif instance.text == '=':  
 try:  
 textinput.text = str(round(eval(textinput.text), 2)) # eval - функция, которая вычисляет значение заданной строки. round - вывод 2-ух цифр после запятой. str - преобразует результат в строку для вывода в textinput  
 except SyntaxError: # если пользователь ввел ничего  
 textinput.hint\_text = 'ВВЕДИТЕ ВЫРАЖЕНИЕ' # изменяем пустой текст для ввода  
 textinput.hint\_text\_color = [1, 0, 0, 1] # красим его в красный  
 textinput.font\_size = 25 # меняем ему размер  
 else: # в любых других случаях  
 textinput.text += instance.text # добавляем к полю ввода текст, который был на кнопке  
  
 for i in range(9, -1, -1): # цикл на добавление кнопок в табличный макет  
 gridlayout.add\_widget(Button(text=str(i),  
 font\_size=30,  
 on\_press=work\_with\_textinput))  
 if i == 7:  
 gridlayout.add\_widget(Button(text='С',  
 font\_size=30,  
 on\_press=work\_with\_textinput))  
 elif i == 4:  
 gridlayout.add\_widget(Button(text='+',  
 font\_size=30,  
 on\_press=work\_with\_textinput))  
 elif i == 1:  
 gridlayout.add\_widget(Button(text='-',  
 font\_size=30,  
 on\_press=work\_with\_textinput))  
 gridlayout.add\_widget(Button(text='=',  
 font\_size=30,  
 on\_press=work\_with\_textinput))  
 elif i == 0:  
 gridlayout.add\_widget(Button(text='\*',  
 font\_size=30,  
 on\_press=work\_with\_textinput))  
 gridlayout.add\_widget(Button(text='/',  
 font\_size=30,  
 on\_press=work\_with\_textinput))  
 boxlayout.add\_widget(textinput) # добавляем поле ввода в общий макет  
 boxlayout.add\_widget(gridlayout) # добавляем таблицу, или же кнопки в общий макет  
 return boxlayout # возвращаем то что хотим видить  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 MyApp().run()

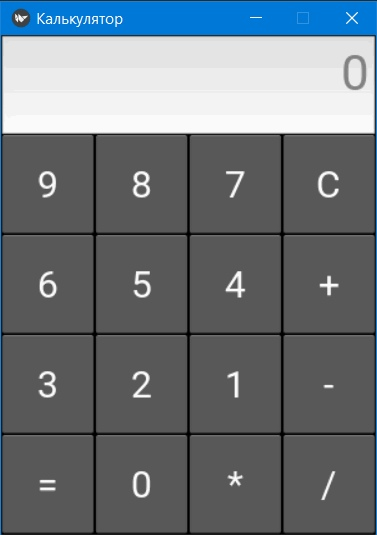


Рисунок 3.2 – Простой калькулятор

Некоторые замечания.

В начале мы создали макет (boxlayout), который располагал добавленные в него виджеты сверху вниз (так как orientation = 'vertical'). В макет поместили поле ввода (textinput) и табличный макет (gridlayout), который затем наполнили кнопками.

К кнопкам привязали функцию work\_with\_textinput, которая считывала значения с кнопки при нажатии на неё. Если это значение было 'C', то поле ввода очищалось.

По нажатии кнопки '=', вызывалась функция eval(), которая рассчитывала ответ выражения, и он выводился в поле ввода.

Задания.

*Вариант 1.*

Конвертер валют. Пользователь вводит сумму в рублях, выбирает валюту, в которую необходимо сконвертировать, получает эквивалент в выбранной валюте.

*Вариант 2.*

Ежедневник. Реализовать возможность создания заметок и вывода их на экран в по дате.

*Вариант 3.*

Календарь. Реализовать возможность выбора года и вывод чисел по дням недели.

*Вариант 4.*

Часы. Реализовать часы с обновлением каждую минуту.

*Вариант 5.*

Будильник. Реализовать добавление будильника. При активации будильника воспроизводить мелодию.

*Вариант 6.*

Секундомер. Реализовать запуск секундомера, круги, остановку и запись результатов в файл.

*Вариант 7.*

Таймер. Реализовать запуск таймера, звуковой сигнал по активации таймера.

*Вариант 8.*

Продвинутый калькулятор. Реализовать калькулятор с возможностью возведения в степень, извлечения корня, добавить число Пи, Эйлера.

*Вариант 9.*

Телефонная книга. Реализовать добавление контакта, редактирование и удаление.

*Вариант 10.*

Блокнот. Реализовать ввод текста с рабочими горячими клавишами (копирование, вставка). Сохранение файла в трех расширениях.

*Вариант 11.*

*Вариант 12.*

*Вариант 13.*

*Вариант 14.*

*Вариант 15.*

*Вариант 16*.

*Вариант 17.*

*Вариант 18.*

*Вариант 19.*

*Вариант 20.*

*Вариант 21.*

*Вариант 22.*

*Вариант 23.*

*Вариант 24.*

*Вариант 25.*

**Лабораторная работа №3**

**Получение навыков по работе с Kivy**

Цель.

Улучшить навыки работы с фреймворком Kivy.

Вот пример простого аудиоплеера, написанного на Python Kivy:

from kivy.config import Config  
  
Config.set('graphics', 'resizable', 0) # делаемым окно не изменяемым для пользователя  
  
from kivy.app import App # окно  
from kivy.core.window import Window #  
from kivy.uix.boxlayout import BoxLayout # размещение по порядку по квадратам  
from kivy.uix.gridlayout import GridLayout # табличное размещение  
from kivy.uix.button import Button # кнопка  
from kivy.uix.label import Label  
from kivy.uix.slider import Slider # слайдер (полоса для прокрутки)  
from kivy.uix.widget import Widget  
from kivy.uix.image import Image  
from kivy.clock import Clock # временем в киви  
from kivy.core.audio import SoundLoader # работа с аудио в киви  
from kivy.uix.textinput import TextInput # поле для ввода  
import os # работа с файлаами/ папками  
from mutagen.mp3 import MP3 # работа с аудио (более оптимизированная)  
import time  
from kivy.uix.scrollview import ScrollView # возможность прокручивать длинные таблицы  
from functools import partial # для скролла  
from kivy.uix.progressbar import ProgressBar # прогресс аудио  
  
  
class MyApp(App): # класс создания главного окна  
 # классовые переменные  
 bl = BoxLayout(orientation='vertical') # главный бокс лайаут  
 sound = SoundLoader.load('1.mp3') # аудио которая сейчас играет  
 time\_of\_song = None # время остановки аудио  
 tempinstance = None # временная кнопка  
 path = None # путь к аудио пользователя  
 pb = ProgressBar(size\_hint=(None, None),  
 size=('500', '60'))  
  
 def \_\_init\_\_(self, \*\*kwargs):  
 super(MyApp, self).\_\_init\_\_(\*\*kwargs)  
 self.title = 'AudioPlayer from Kivy : Not Play' # переименовывание окна

def on\_stop(self):  
 self.sound.stop()

def build(self): # главная функция окна, которая запускается по умолчанию  
 workspace = BoxLayout(orientation='vertical',  
 padding=[260, 285, 260, 285]) # делаем панель посередине  
 bl = BoxLayout(orientation='horizontal') # создаем там лайаут для кнопки и поля ввода  
 workspace.add\_widget(bl)  
  
 text\_input = TextInput(text='', # поле ввода  
 hint\_text='Введите директорию C музыкой >', # текст. который показывается если поле пусто  
 multiline=False, # для работы нажатия Enter  
 on\_text\_validate=self.read\_the\_dir,) # функция которая будет вызыватся после нажатия enter  
 bl.add\_widget(text\_input) # добавляем поле ввода в центральный лайаут  
  
 def on\_press(instance): # после нажатия кнопки  
 self.read\_the\_dir(text\_input)  
  
 btn = Button(text='Ok',  
 size\_hint=[0.1, 1],  
 on\_press=on\_press,)  
 bl.add\_widget(btn)  
 self.bl.add\_widget(workspace)  
 return self.bl  
  
 def read\_the\_dir(self, instance): # чтение введенной директории на наличие mp3 файлов  
 self.path = instance.text  
 try: # если вдруг пользователь указал неверную папку  
 ls\_of\_all\_files = os.listdir(path=self.path)  
 except FileNotFoundError: # очищаем поле и выходим из функции  
 instance.text = ''  
 instance.hint\_text\_color = [1, 0, 0, 1]  
 return  
 ls\_of\_music = []  
 for i in ls\_of\_all\_files:  
 if i.find('.mp3') > 0:  
 ls\_of\_music.append(i) # находим всем mp3 и вводим их в список  
 if len(ls\_of\_music) == 0: # если ни нашли не одной аудио, выходим  
 instance.text = ''  
 instance.hint\_text\_color = [1, 0, 0, 1]  
 return  
 ls\_of\_music.sort()  
 # если же нашли, то реверсируем список, чтоб был по алфавиту  
 self.bl.clear\_widgets()  
 self.full\_list(ls\_of\_music)  
  
 def full\_list(self, ls\_of\_music):  
 workspace = BoxLayout(orientation='horizontal', # рабочая область для gridlayout'a  
 padding=[10, 10, 10, 0])  
 scroll = ScrollView(size\_hint=(None, None), # возможность скроллить  
 size=(755, Window.height),  
 )  
 workspace.add\_widget(scroll)  
 gl = GridLayout(cols=2, # создаем саму таблицу, в 2 столбика  
 padding = [0, 100, 0, 0]

row\_force\_default=True,  
 row\_default\_height=40,  
 width=750,  
 height=40,  
 size\_hint=(None, None),  
 )  
 gl.bind(minimum\_height=gl.setter('height'))  
 scroll.add\_widget(gl)  
  
 self.bl.add\_widget(workspace)  
  
 for i in ls\_of\_music:  
 name\_of\_song = i # первый столбец - названи песни  
 gl.add\_widget(Button(text=name\_of\_song,  
 size\_hint\_x=None,  
 width=650,  
 on\_press=self.play\_music,  
 ))  
 audio = MP3(self.path + '\\' + name\_of\_song) # используем для получения длительности песни  
 length = time.strftime('%M:%S', time.gmtime(audio.info.length)) # переводим uix время в нормалное  
 gl.add\_widget(Button(text=length)) # второй столбик - длительность песни  
  
 sl = Slider(orientation='vertical', # создаем слайдер. или же панель для прокрутки списка  
 min=0, max=1, value=25,)  
  
 sl.bind(value=partial(self.scroll\_change, scroll)) # привязываем скролл к слайдеру  
 scroll.bind(scroll\_y=partial(self.slider\_change, sl))  
  
 workspace.add\_widget(sl)  
 self.player() # если все удачноп прошло, загружаем плеер  
 return self.bl  
  
 def play\_music(self, instance):  
 if self.sound.state == "play": # если музыка играет, останавливаем её, выгружаем и загружаем новую  
 self.tempinstance.background\_color = [1, 1, 1, 1] # красим поле с воспроизводимой музыкой  
 self.sound.unload() # выгружаем старую аудио  
 self.tempinstance = instance # запоминаем текущую кнопку, чтоб потом перекрасить обратно  
 instance.background\_color = [1, 0, 0, 1] # красим текущую кнопку  
 self.title = 'AudioPlayer from Kivy : ' + instance.text # переименовываем окно  
 self.sound = SoundLoader.load(self.path + '\\' + instance.text) # загружаем новую песню  
 self.sound.volume = 0.2 # ставим громкость воспроизведения по умолчанию  
 audio = MP3(self.path + '\\' + instance.text) # считываем длительность песни  
 self.pb.max = audio.info.length # привязываем длительность прогресс бара к длительности песни  
 self.sound.play() # запускаем песню

Clock.schedule\_once(self.pbprogress)  
 # вызывает функцию на изменения прогресс бара

def player(self):  
 bl = BoxLayout(orientation='horizontal',  
 size\_hint\_x=1,  
 size\_hint\_y=None)  
 self.bl.add\_widget(bl) # создаем и добавляем изначальный лайаут. в котором будет прогресс бар и кнопки  
  
 bl.add\_widget(self.pb) # добавляем прогресс бар в плеер  
  
  
 def volume\_plus(instance): # функция на повышение громкости  
 if self.sound.volume < 0.9:  
 self.sound.volume += 0.1  
  
 btn\_volumeplus = Button(text='Vol+',  
 on\_press=volume\_plus,  
 size\_hint=(None, None),  
 size=('50', '60'))  
 bl.add\_widget(btn\_volumeplus)  
  
 def volume\_minus(instance): # функция на понижение громкости  
 if self.sound.volume > 0.2:  
 self.sound.volume -= 0.1  
  
 btn\_volumemin = Button(text='Vol-',  
 on\_press=volume\_minus,  
 size\_hint=(None, None),  
 size=('50', '60'))  
 bl.add\_widget(btn\_volumemin)  
  
 def pause(instance): # функция остановки на паузу  
 if self.sound.state == 'play':  
 self.time\_of\_song = self.sound.get\_pos() # записываем в переменную текущую позицию аудио  
 self.tempinstance.background\_color = [1, 1, 1, 1]  
 self.sound.stop()  
 else:  
 self.sound.play()  
 self.tempinstance.background\_color = [1, 0, 0, 1]  
 while self.sound.get\_pos() <= self.time\_of\_song:  
 self.sound.seek(self.time\_of\_song) # воспроизводим с прошлой позиции  
 time.sleep(.1)

Clock.schedule\_once(self.pbprogress)

btn\_pause = Button(text='Pause',  
 on\_press=pause,  
 size\_hint=(None, None),  
 size=('60', '60'))  
 bl.add\_widget(btn\_pause)  
 return self.bl  
  
 def scroll\_change(self, scroll, instance, value): # изменяем позицию скролла  
 scroll.scroll\_y = value  
  
 def slider\_change(self, sl, instance, value): # изменяем позицию слайдера  
 if value >= 0:  
 sl.value = value  
  
 def pbprogress(self, instance): # обновляем прогресс бар  
 if self.sound.state == 'play':  
 self.pb.value = self.sound.get\_pos()

Clock.schedule\_once(self.pbprogress)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 MyApp().run()

Задания.

*Вариант 1.*

Написать программу для просмотра изображений нескольких форматов. Создание меню по управлению показа изображений (следующее, предыдущее изображение), слайдшоу из нескольких изображений.

*Вариант 2.*

Написать программу для просмотра видео нескольких форматов, создание меню по управлению показа видео (пауза, продолжить, стоп, следующее видео, предыдущее видео).

*Вариант 3.*

Реализация русско-английского и англо-русского словаря. Пользователь вводит слово, ему выводится несколько вариантов перевода.

*Вариант 4.*

Реализовать приложение часы с будильником, секундомером и таймером.

*Вариант 5.*

Реализовать простой проводник. Добавить возможности перехода по директориям, создание и удаление файлов/папок, их переименовывание.

*Вариант 6.*

Реализовать поиск файла по названию в выбранной директории.

*Вариант 7.*

Реализовать простейший парсер, считывающий погоду с сайта. Вывести результат в окно.

*Вариант 8.*

Реализовать простейший парсер, считывающий курсы валют, с официального источника.

*Вариант 9.*

Реализовать программу для смены обоев на рабочем столе. Обои будут скачиваться из источника в интернете.

*Вариант 10.*

Реализовать простейшую программу для рисования граффити. Добавить сохранение изображения в файл и очищение экрана.

Список литературы

1. Официальный сайт языка программирования Python – <https://www.python.org/>
2. Официальный сайт программы от JetBrains , PyCharm - <https://www.jetbrains.com/pycharm/>
3. Стайлгайд PEP - <https://www.python.org/dev/peps/pep-0008/>
4. Swaroop C. H. “A Byte of Python” - <http://wombat.org.ua/AByteOfPython/AByteofPythonRussian-2.01.pdf>
5. IBM. Программирование на Python – <https://www.ibm.com/developerworks/ru/library/l-python_part_1/index.html>
6. Официальный сайт Kivy - <https://kivy.org/doc/stable/api-kivy.html>

Список рекомендуемой литературы

1. Бхаргава А. “Грокаем Алгоритмы”
2. Дэн Бейдер “Чистый Python [Тонкости программирования для профи]”
3. Лучано Рамальо “Python. К вершинам мастерства”
4. Онлайн курсы <https://www.udacity.com/course/introduction-to-python--ud1110>