МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ
Колледж Пятигорского института (филиал) СКФУ

ОП. 06 ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

Специальности СПО

09.02.11 Разработка и управление программным обеспечением

Квалификация: программист

Пятигорск 2026

Методические указания для лабораторных работ по дисциплине ОП.06 «Основы алгоритмизации и программирования» составлены в соответствии с требованиями ФГОС СПО к подготовке выпуска для получения квалификации. Предназначены для студентов, обучающихся по специальности 09.02.11 Разработка и управление программным обеспечением.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Программа Основы алгоритмизации и программирования предусматривает изучение языков программирования.

При изучении предмета следует соблюдать единство терминологии и обозначения в соответствии с действующими стандартами, Международной системной единицы (СИ).

В результате освоения учебной дисциплины обучающийся должен

уметь

- разрабатывать и анализировать алгоритмы для решения поставленных задач;
- определять сложность алгоритмов;
- реализовывать типовые алгоритмы в виде программ на актуальных языках программирования;
- использовать средства проектирования для создания и графического отображения алгоритмов;
- оформлять код программ в соответствии со стандартом кодирования;
- выполнять проверку, отладку кода программы.

знать

- понятие алгоритмизации, свойства алгоритмов, общие принципы построения алгоритмов, основные алгоритмические конструкции;
- классификация языков программирования;
- понятие системы программирования;
- основные элементы языка, структура программы;
- методы реализации типовых алгоритмов;
- операторы и операции, управляющие структуры, структуры данных, классы памяти;
- понятие подпрограммы, библиотеки подпрограмм;
- объектно-ориентированная модель программирования, основные принципы объектно-ориентированного программирования на примере алгоритмического языка: понятие классов и объектов, их свойств и методов, инкапсуляции и полиморфизма, наследования и переопределения.

Лабораторная работа 1. Составление и оформление блок-схем простых алгоритмов.

Тема 1. Понятие алгоритма. Способы описания алгоритма. Базовые алгоритмические конструкции.

Цель: изучить способы построения блок-схем и написание программного кода.

Оборудование: ПК, ОС Windows/Linux, Python 3, среда разработки, методические указания.

Ход работы:

Блок-схемой будем называть такое графическое представление алгоритма, когда отдельные действия (или команды) представляются в виде геометрических фигур – *блоков*. Внутри блоков указывается информация о действиях, подлежащих выполнению. Связь между блоками изображают с помощью линий, называемых *линиями связи*, обозначающих передачу управления.

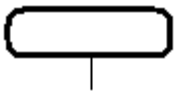
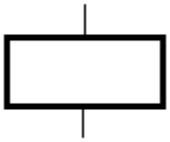
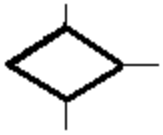
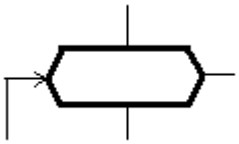
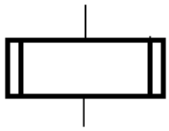
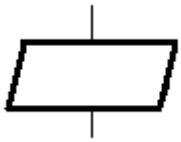
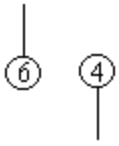
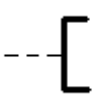
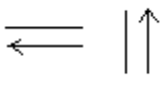
Существует Государственный стандарт, определяющий правила создания блок-схем. Конфигурация блоков, а также порядок графического оформления блок-схем регламентированы ГОСТ 19.701-90 "Схемы алгоритмов и программ".

Правила составления блок-схем:

1. Каждая блок-схема должна иметь блок «*Начало*» и один блок «*Конец*».
2. «*Начало*» должно быть соединено с блоком «*Конец*» линиями потока по каждой из имеющихся на блок-схеме ветвей.
3. В блок-схеме не должно быть блоков, кроме блока «*Конец*», из которых не выходит линия потока, равно как и блоков, из которых управление передается «в никуда».
4. Блоки должны быть пронумерованы. *Нумерация* блоков осуществляется сверху вниз и слева направо, номер блока ставится вверху слева, в разрыве его начертания.
5. Блоки связываются между собой линиями потока, определяющими последовательность выполнения блоков. Линии потоков должны идти параллельно границам листа. *Если линии идут* ***справа налево*** *или* ***снизу вверх****, то стрелки в конце линии обязательны*, в противном случае их можно не ставить.
6. По отношению к блокам линии могут быть *входящими* и *выходящими*. Одна и та же линия потока является выходящей для одного блока и входящей для другого.
7. От блока «*Начало*» в отличие от всех остальных блоков линия потока только выходит, так как этот блок – первый в блок-схеме.
8. Блок «*Конец*» имеет только вход, так как это последний блок в блок-схеме.
9. Для простоты чтения желательно, чтобы линия потока входила в блок «Процесс» сверху, а выходила снизу.
10. Чтобы не загромождать блок-схему сложными пересекающимися линиями, линии потока можно разрывать. При этом в месте разрыва ставятся *соединители*, внутри которых указываются номера соединяемых

блоков. В блок-схеме не должно быть разрывов, не помеченных соединителями.

11. Чтобы не загромождать блок, можно информацию о данных, об обозначениях переменных и т.п. размещать в *комментариях* к блоку.

Название блока	**Обозначение блока**	**Назначение блока**
1	**2**	**3**
Терминатор		Начало/Конец программы или подпрограммы
Процесс		Обработка данных (вычислительное действие или последовательность вычислительных действий)
Решение		Ветвление, выбор, проверка условия. В блоке указывается условие или вопрос, который определяет дальнейшее направление выполнения алгоритма
Подготовка		Заголовок счетного цикла
Предопределенный процесс		Обращение к процедуре
Данные		Ввод/Вывод данных
Соединитель		Маркировка разрыва линии потока
Комментарий		Используется для размещения пояснений к действиям
Горизонтальные и вертикальные потоки		Линии связей между блоками, направление потоков

Задание:

1) Создать алгоритм с помощью блок-схем и написать программу для вычисления суммы двух чисел.

2) Создать алгоритм с помощью блок-схем и написать программу для вычисления разности двух чисел.

3) Создать алгоритм с помощью блок-схем и написать программу для вычисления площади квадрата.

4) Создать алгоритм с помощью блок-схем и написать программу для вычисления объема куба.

5) Создать алгоритм с помощью блок-схем и написать программу для сравнения двух чисел между собой, для определения большего.

6) Создать алгоритм с помощью блок-схем и написать программу для сравнения двух чисел между собой, для определения меньшего.

Вопросы для самоконтроля

1. Понятие алгоритма.
2. Свойства и виды алгоритмов.
3. Способы описания алгоритмов: псевдокоды.
4. Правила составления блок-схем.
5. Стандарты графического оформления алгоритмов.
6. Линейные алгоритмы.
7. Разветвляющиеся алгоритмы.
8. Циклические алгоритмы.
9. Критерии «хорошего» алгоритма.

Лабораторная работа 2. Проектирование и оформление алгоритмов сортировки и поиска.

Тема 2. Основные методы и этапы проектирования алгоритмов. Эффективность и сложность алгоритма.

Цель: изучить алгоритмы сортировки одномерных массивов.

Оборудование: ПК, ОС Windows/Linux, Python 3, среда разработки, методические указания.

Ход работы:

Сортировка одномерных массивов.

В практике программиста часто случается так, что нужно разместить какие-либо данные в определенном порядке. В Паскале для таких случаев предусмотрена сортировка. Существует два основных алгоритма сортировки. Первый из них - **метод прямого выбора**. Ее смысл заключается в том, что за счет вложенности циклов каждый элемент массива сравнивается с остальными.

То есть, если у нас 10 чисел, то сначала первое из них будет сравниваться до тех пор, пока не будет найдено другой, например, большее его (если мы сортируем просто по возрастанию). Далее так же будет сравниваться 2, 3, 4 … элементы с последующими, но не с теми, которые уже отсортированы.

Прелесть этого вида сортировки заключается в том, что она очень проста для начинающего программиста, и все обычно начинают именно с неё. Давайте же рассмотрим пример. Пусть нам дан массив из 20 элементом и нам нужно отсортировать по убыванию.

```
import random

# Инициализируем список из 20 случайных целых чисел от 0 до 99
mas = [random.randint(0, 99) for _ in range(20)]

print("Массив до сортировки:")
print(*mas)

# Пузырьковая сортировка по убыванию
n = len(mas)
for i in range(n - 1):
    for j in range(i + 1, n):
        if mas[i] < mas[j]:
            # Меняем элементы местами
            mas[i], mas[j] = mas[j], mas[i]

print("\nМассив после сортировки:")
print(*mas)
```

Обратите внимание на 2 первые строки после Begin. Здесь мы вызываем процедуру генерации случайных чисел. То есть просто заполняем наш массив числами от 1 до 100. Далее в цикле выводится для пользователя первоначальный массив.

Далее следует сам алгоритм сортировки, для новичков объясним, что это 2 цикла for, первый из них, в данной случае, идет с самого начала и до N-1 (1 первого до предпоследнего элемента).

Далее вложенный for, где используется уже другой счетчик j. Он изменяется от текущего i-того, увеличенного на 1 и до конца. И далее мы проверяем 2 элемента массива в условии и, если все нас устраивает, то производим обмен переменными. В последнем цикле мы выводим отсортированный массив.

Заметим, что первоначальный массив при сортировке не сохраняется. Кроме того, хочу вам показать, как можно сделать так, чтобы первая половина массива была отсортирована по возрастанию, а вторая - по убыванию.

```
import random

# Генерируем массив из 10 случайных чисел от 0 до 100
mas = [random.randint(0, 100) for _ in range(10)]

# Выводим исходный массив
print("Массив до сортировки:", *mas)

# Сортируем первую половину массива по возрастанию
for i in range(len(mas)//2):  # От 0 до середины (4 включительно)
    for j in range(i + 1, len(mas)//2):  # Следующие элементы первой половины
        if mas[i] > mas[j]:  # Проверяем условие для сортировки по возрастанию
            mas[i], mas[j] = mas[j], mas[i]  # Меняем местами элементы

# Сортируем вторую половину массива по убыванию
for i in range(len(mas)//2, len(mas)-1):  # Начинаем с второй половины
    for j in range(i + 1, len(mas)):  # Остальные элементы во второй половине
        if mas[i] < mas[j]:  # Условие для сортировки по убыванию
            mas[i], mas[j] = mas[j], mas[i]  # Меняем местами элементы

# Выводим отсортированный массив
        print("Массив после сортировки:", *mas)
```

Надеюсь, что вы заметили, что здесь 2 алгоритма сортировки, первый идет с первого элемента по 5, а второй - с 5 по 10. Они отличаются только условиями, в этом и заключается этот нехитрый способ.

Как видите, здесь нет ничего особенного. И еще один пример по сортировке выбором. Пусть нудно отсортировать массив таким образом, чтобы числа в нем располагались в порядке возрастания своих последних разрядов. Допустим, если у нас массив 17 23 18 70, то после сортировки он должен выглядеть так: 70 23 17 18. Вот код этой программы.

```
import random

# Генерация массива из 10 случайных чисел от 0 до 99
arr = [random.randint(0, 99) for _ in range(10)]

# Вывод массива до сортировки
```

```
print("Массив до сортировки:", arr)

# Сортировка массива по остатку от деления на 10
for i in range(len(arr)):
    for j in range(i + 1, len(arr)):
        if arr[i] % 10 > arr[j] % 10:
            arr[i], arr[j] = arr[j], arr[i]

# Вывод отсортированного массива
        print("Массив после сортировки:", arr)
```

Эта программа схожа с первой, их отличие лишь в том, что разное условие для перестановки элементов массива. В первой это «if mas[i] < mas[j] then», а во втрой «if a[i] mod 10 > a[j] mod 10 then» То есть мы располагаем наши элементы в таком необычном порядке благодаря тому, что просто проверяем остатки их деления на 10, то есть и сравниваем их последние цифры.

Теперь подробнее о **сортировке пузырьком**. Его сущность заключается в том, что мы сравниваем соседние элементы парами: 1 и 2, 2 и 3, 3 и 4, 4 т.д. И если наш элемент удовлетворяет условию сортировки, то мы его проталкиваем в конец массива, он как бы всплывает, прямо как «пузырек». От этого и название этого алгоритма. Вот пример программы с этим алгоритмом.

```
import random

# Создание массива из 20 случайных чисел от 0 до 99
mas = [random.randint(0, 99) for _ in range(20)]

# Вывод массива до сортировки
print("Массив до сортировки:", mas)

# Применение пузырьковой сортировки
for i in range(len(mas) - 1):
    for j in range(len(mas) - 1 - i):
        if mas[j] > mas[j + 1]:
            # Меняем местами два соседних элемента
            mas[j], mas[j + 1] = mas[j + 1], mas[j]

# Вывод отсортированного массива
        print("Отсортированный массив:", mas)
```

Ее алгоритм чем-то похож на сортировку выбором. Здесь так же 2 цикла for, но второй цикл идет с 1 до 20 - i.

Кроме того, существует второй, не менее известный алгоритм сортировки «пузырьком» с флагом. Флаг - это переменная типа boolean. Из-за этого меняется структура алгоритма, но сущность остается прежней. Вот он.

```
import random
```

```
# Создаем массив из 20 случайных чисел от 0 до 99
a = [random.randint(0, 99) for _ in range(20)]

# Выводим массив до сортировки
print("Массив до сортировки:", a)

# Начало сортировки
i = 0
while True:
    i += 1
    flag = False
    # Проходим по массиву сверху вниз
    for j in range(19, i - 1, -1):
        if a[j] < a[j + 1]:
            # Меняем элементы местами
            a[j], a[j + 1] = a[j + 1], a[j]
            flag = True

    # Если никаких изменений не произошло, прекращаем сортировку
    if not flag:
        break

# Выводим отсортированный массив
        print("Массив после сортировки:", a)
```

Этот алгоритм наиболее сложный для запоминания, но уверяю Вас, что его и не нужно зазубривать, а понимать, что за чем идет. Здесь используется внешний цикл repeat … until not flag; Он существует до тех пор, пока флаг не окажется истинным, то есть flag := true; А это может произойти, если наше условие сортировки выполняется. И наш элемент тоже всплывает, как пузырек. Итак, надеюсь, что вы поняли, в чем отличие этих 2 алгоритмов сортировок и теперь сможете отсортировать любой массив так, как Вам нужно.

Последовательный поиск элемента в массиве

Линейный или последовательный поиск - самый простой из алгоритмов поиска элемента в массиве.

Алгоритм заключается в обходе всех элементов массива, как правило, слева на право, и сравнения их с искомым значением. Если значения элемента и ключа совпадают, то поиск возвращает индекс элемента.

По скольку линейный алгоритм, обходит массив последовательно, он очень медленный.

Тем не менее этот метод используется для поиска:

- на небольших массивах данных;
- в потоковой обработке данных;
- поиске минимального и максимального значения массива;
- на одиночных неупорядоченных больших массивах.

Программа для последовательного поиска элемента массива:

```
import random

def linear_search(array, target):
    """Последовательный поиск элемента в списке"""
    for idx, value in enumerate(array):
        if value == target:
            return idx  # Возвращаем индекс найденного элемента
    return -1  # Элемента нет в списке

# Длина массива
ARRAY_LENGTH = 10

# Генерируем случайный массив
random.seed()  # Инициализируем генератор случайных чисел
input_array = [random.randint(0, 99) for _ in range(ARRAY_LENGTH)]

# Выводим исходный массив
print("Исходный массив:", input_array)

# Получаем искомое значение от пользователя
key = int(input("Искомое значение: "))

# Осуществляем поиск
index = linear_search(input_array, key)

# Выводим результат
if index != -1:
    print(f'Индекс элемента в массиве: {index}')
else:
            print('Элемент не найден.')
```

Функция для линейного поиска с использованием цикла for

В функции используется цикл while do, можно легко модифицировать программу использовав цикл for.

```
def linear_search(array, target):
    """
    Линейный поиск элемента в массиве.
    Возвращает индекс найденного элемента или -1, если элемент не найден.
    """
    for i in range(len(array)):
        if array[i] == target:
            return i  # Вернуть индекс найденного элемента
    return -1  # Если элемент не найден
my_list = [12, 34, 56, 78, 90]
result = linear_search(my_list, 56)
if result != -1:
    print(f"Элемент найден на позиции {result}")
else:
```

```
print("Элемент не найден")
```

Максимальный элемент массива

Алгоритм поиска **максимального элемента** неупорядоченного массива заключается в следующем:

- сначала мы предполагаем, что наибольший элемент находиться в начале массива;
- сохраняем значение первого элемента в переменной;
- затем мы сравниваем его с другими элементами массива один за другим, если какой-либо элемент больше, чем наш предполагаемый максимум, то обновляется значение переменной;
- после обхода всего массива, возвращаем максимальный элемент.

Код программы для поиска максимального элемента массива:

```
import random

# Задали длину массива
ARRAY_LENGTH = 10

# Генерация случайного массива из 10 элементов (числа от 0 до 99)
input_array = [random.randint(0, 99) for _ in range(ARRAY_LENGTH)]

# Выводим исходный массив
print("Исходный массив:", input_array)

# Нахождение максимального элемента
# Изначально считаем первым элементом максимальный
maximum = input_array[0]

# Последовательно проходим по остальным элементам массива
for element in input_array[1:]:
    if element > maximum:
        maximum = element

# Выводим максимальное значение
        print("Максимальный элемент массива:", maximum)
```

Рекурсивный алгоритм нахождения максимального значения элемента массива

Найти максимальный элемент массива, можно также - рекурсивно. Реализация метода немного сложнее предыдущего, однако полезна для изучения принципов рекурсивных вызовов функций.

```
import random



# Размер массива

ARRAY_LEN = 10
```

```
# Случайный массив из 10 элементов
input_arr = [random.randint(0, 99) for _ in range(ARRAY_LEN)]

# Рекурсивная функция поиска максимального элемента
def find_max_element(current_max, index):
    # Базовый случай: дошли до конца массива
    if index >= ARRAY_LEN:
        return current_max
    else:
        # Если текущий элемент больше текущего максимума,
        # обновляем максимум
        if input_arr[index] > current_max:
            current_max = input_arr[index]

        # Рекурсивный вызов для следующего элемента
        return find_max_element(current_max, index + 1)

# Основной блок программы
print("Исходный массив:", input_arr)

# Начальное значение максимума берем как первый элемент массива
initial_max = input_arr[0]

# Запускаем рекурсивный поиск начиная со второго элемента
final_max = find_max_element(initial_max, 1)

# Выводим результат
```

```
print("Наибольший элемент =", final_max)
```

Минимальный элемент массива

Найти минимальный элемент массива очень просто. Если это упорядоченный массив, то достаточно вернуть первое или последнее значение, в зависимости от того, как отсортированы данные, от наименьшего к наибольшему или от наибольших к наименьшим. Это очень простая задача.

В случае с неотсортированным массивом, задача поиска минимального значения элемента сводиться к полному обходу всех элементов и выбора из них - минимума.

Код программы для поиска минимального, по значению, элемента неупорядоченного массива

```
import random

# длина массива
ARRAY_LENGTH = 10

# создание массива из 10 случайных чисел от 0 до 99
input_array = [random.randint(0, 99) for _ in range(ARRAY_LENGTH)]

# вывод исходного массива
print("Исходный массив:", input_array)

# поиск минимального элемента
minimum = input_array[0]  # считаем первый элемент минимальным

for num in input_array[1:]:  # начинаем проверять с 2-го элемента
   if minimum > num:
      minimum = num  # обновляем минимальное значение

# вывод минимального элемента
      print("Минимальный элемент массива:", minimum)
```

Найти *минимальное значение*, можно также, с использованием рекурсивного алгоритма.

Рекурсивный алгоритм поиска минимального элемента в одномерном массиве

```
import random


# Константа, определяющая длину массива

ARRAY_LEN = 10


# Функция для рекурсивного поиска минимального элемента
```

```
def min_element(current_min, index):
    # Базовый случай: достигли конца массива
    if index > ARRAY_LEN:
        return current_min
    else:
        # Если текущий элемент меньше текущего минимума, обновляем минимум
        if input_arr[index - 1] < current_min:
            current_min = input_arr[index - 1]
        # Рекурсивный вызов для следующего элемента
        return min_element(current_min, index + 1)

# Генератор случайного массива
input_arr = [random.randint(0, 99) for _ in range(ARRAY_LEN)]

# Вывод исходного массива
print("Исходные данные:", input_arr)

# Поиск минимального элемента, стартуя с первого элемента
current_min = input_arr[0]
final_min = min_element(current_min, 2)

# Вывод минимального элемента
print("Минимальный элемент ==", final_min)
```

Бинарный поиск элемента в массиве

Бинарный поиск (binary search) - алгоритм поиска индекса элемента в упорядоченном массиве, в нем используется деление массива на половины, по это й причине алгоритм называют **методом деления пополам**.

Метод бинарного поиска достаточно прост для понимания, в то же время он очень эффективен. Поскольку на каждой итерации количество элементов в рабочей области массива уменьшается вдвое.

Описание алгоритма бинарного поиска

- определяем значение элемента в средине рабочей области массива и сравниваем его с искомым;
- если они равны, выводим значение;
- если значение середины больше искомого, то поиск продолжается в первой половине, иначе во второй;
- проверяем не сошлись ли границы рабочей области, если да - искомого значения нет, нет - переходим на первый шаг.

Рекурсивная реализация бинарного поиска

```
# Длина массива
ARRAY_LENGTH = 15

# Функция бинарного поиска
def bin_search(key, left_idx, right_idx):
    if left_idx > right_idx:
        return -1  # Значение не найдено
    else:
        mid_idx = (left_idx + right_idx) // 2
        if sorted_array[mid_idx] == key:
            return mid_idx  # Нашли значение
        elif sorted_array[mid_idx] > key:
            return bin_search(key, left_idx, mid_idx - 1)  # Продолжаем искать слева
        else:
            return bin_search(key, mid_idx + 1, right_idx)  # Продолжаем искать справа

# Создаем отсортированный массив четных чисел
sorted_array = list(range(2, 2 * (ARRAY_LENGTH + 1), 2))

# Выводим исходный массив
print("Исходный массив:", sorted_array)

# Просим ввести искомое значение
search_key = int(input("Введите значение искомого элемента: "))

# Запускаем бинарный поиск
index = bin_search(search_key, 0, len(sorted_array) - 1)

# Выводим результат
if index == -1:
    print("Элемент не найден.")
else:
    print("Индекс элемента в массиве:", index)
```

Итеративная реализация алгоритма бинарного поиска

```
# Количество элементов в массиве
ELEMENTS_COUNT = 15
```

```
# Функция бинарного поиска
def binary_search(key):
    first_index = 0          # Левая граница диапазона поиска
    last_index = ELEMENTS_COUNT - 1  # Правая граница диапазона поиска

    while first_index <= last_index:
        mid_index = (first_index + last_index) // 2  # Средний индекс

        if sorted_arr[mid_index] == key:
            return mid_index  # Элемент найден

        elif sorted_arr[mid_index] > key:
            last_index = mid_index - 1  # Смещаемся левее

        else:
            first_index = mid_index + 1  # Смещаемся правее

    return -1  # Элемент не найден


# Формирование отсортированного массива (каждый следующий элемент на 3 больше предыдущего)
sorted_arr = [(i + 1) * 3 for i in range(ELEMENTS_COUNT)]

# Выводим исходный массив
print("Исходный массив:", sorted_arr)

# Запрашиваем искомое значение
key = int(input("Введите искомое значение: "))

# Выполняем бинарный поиск
result = binary_search(key)

# Выводим результат
if result == -1:
    print("Поиск завершен, элемент не найден.")
else:
        print("Индекс элемента в массиве:", result)
```

Вопросы для самоконтроля

1. Алгоритмы поиска.
2. Алгоритмы сортировки.
3. Вложенные циклы.
4. Вспомогательные алгоритмы.
5. Различные комбинации алгоритмических конструкций.
6. Алгоритм Евклида.
7. Декомпозиция алгоритма.

8. Основные методы и этапы проектирования алгоритмов.
9. Нисходящее и восходящее проектирование.
10. Структурное и модульное программирование

Лабораторная работа 3. Проектирование и оформление сложных алгоритмов.

Тема 3. Алгоритмы поиска и сортировки. Различные комбинации алгоритмических конструкций.

Цель: изучить символьный алгоритм и составной алгоритм.

Оборудование: ПК, ОС Windows/Linux, Python 3, среда разработки, методические указания.

Ход работы:

Тип данных char.

В большинстве применений компьютера алфавитно-цифровая информация используется на ряду числовой информацией, прежде чем мы с Вами сможем написать программу, которая манипулирует алфавитно-цифровыми знаками - то есть литерами, нам потребуется тип данных для их представлений.

В языке python для этого существует тип данных char, также как переменная типа integer может хранить одно число, так и переменная типа char может хранить один символ.

Давайте рассмотрим какие значения может принимать переменная типа char в программе, на не большом примере:

```
# Символические переменные в Python представлены обычными строковыми типами.
alpha = 'p'
alpha = '+'  # Присвоение нового значения
alpha = '2'  # Еще одно изменение
alpha = ' '

# Ждем нажатия Enter для продолжения исполнения программы
input()
```

Переменная типа char может принимать значения в виде буквы, и все значения обязательно заключать в одинарные кавычки, тогда программа посчитает этот как символ.

Также переменная может принимать значение в виде знаков - +, -, =, и т.д.

В виде цифры - 1, 2, 3, и т.д.

И также хотим заметить, что символ два не является числом(цифрой), которая может учавствовать в арифметических операциях, а это уже просто символ.

И в переменной может содержаться пробел, хотя мы его и не видим на экране, но всё же это есть символ - значение типа char.

Вы можете увидеть все символы во Free Python в таблице кодов, перейдя в меню - Tools->Ascii table.

Теперь давайте напишем простую программу, которая запрашивает ввод двух литер, и сравнивает их. Как можно сравить литеры? - просто, каждая литера имеет свой номер, если номер одной литеры больше другой, то первая литера естественно больше:

```
# Ввод двух символов без пробела
letters = input('Введите две литеры без пробела - ')
```

```
# Проверка введённых символов
if letters[0] < letters[1]:
   print('Первая литера меньше второй')
elif letters[0] == letters[1]:
   print('Первая литера равна второй')
else:
   print('Первая литера больше второй')

# Ожидание нажатия Enter для завершения программы
input()
```

Как Вы уже должны были заметить литеры вводятся не через пробел, так как он тоже считается литерой.

Также для типа char есть специальные функции, а именно:

Succ() - возвращает следующий символ;

Pred() - возвращает предыдущий символ;

Chr() - возвращает значение кода литеры;

Ord() - возвращает значение литеры по коду;

В первом случае функция будет выводить следующий символ, после символа, который мы дадим ей на обработку, во втором случае всё также, только функция будет возвращать предыдущий символ. Далее идёт функция, которая будет выводить номер литеры данной ей на обработку. И последняя функция по заданному ей номеру литеры, будет выводить саму литеру.

Составное условие.

В этом уроке мы с Вами рассмотрим конструкцию условия, в которой мы будем проверять сразу несколько совпадений - например равно ли первое число второму и второе третьему. Для такой проверки используются зарезервированные слова, которые дают понять программе, что дальше будет ещё одно условие, но в этом случаем будет истина, если оба условия верны, но также можно проверять их не зависимо друг от друга.

Давайте рассмотрим на примере использование сразу несколько условий.

```
# Считывание трех чисел, введенных через пробел
numbers = input("Введите три числа через пробел - ").split()

# Приведение чисел к типу int
num1, num2, num3 = map(int, numbers)

# Проверка на равенство
if num1 == num2 == num3:
   print("Все числа равны!")
else:
   print("Числа не равны!")

# Пауза перед закрытием окна (ожидаем Enter)
input()
```

Мы совместили два условия при помощи команды and(и), также можно было добавить ещё условий. И ещё можно проверять на правильность одно условие из двух, например:

```
# Чтение трёх чисел, введённых через пробел
nums = input("Введите три числа через пробел - ").strip().split()

# Преобразование введённых строковых значений в целые числа
num1, num2, num3 = map(int, nums)

# Проверка наличия равных пар чисел
if (num1 == num2 or num2 == num3 or num1 == num3):
   print("Два числа равны!")
else:
   print("Числа не равны!")

# Пауза перед выходом из программы (чтобы окно не закрывалось автоматически)
input()
```

Наше условие будет проверять первое условие, если нет, то второе, а затем третье, пока не будет совпадений, если нет то возьмёт иначе. Мы использовали команду or(или).

Вопросы для самоконтроля

1. Алгоритмы поиска.
2. Алгоритмы сортировки.
3. Вложенные циклы.
4. Вспомогательные алгоритмы.
5. Различные комбинации алгоритмических конструкций.
6. Алгоритм Евклида.
7. Декомпозиция алгоритма.

Лабораторная работа 4. Изучение инструментария среды программирования на Python. Подготовка структуры программы в среде программирования.

Тема 4. Классификация и генеалогия актуальных языков программирования. Основные элементы языка.

Цель: изучить элементы интерфейса современной IDE (VS Code) и основы создания GUI-приложений на Python с использованием библиотеки Flet.

Оборудование: ПК, ОС Windows/Linux/macOS, интерпретатор Python, текстовый редактор (рекомендуется VS Code), доступ в интернет, методические указания.

Ход работы:

Введение в Python и Flet

Python — это высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Он является кроссплатформенным, то есть программы на Python работают на Windows, Linux, macOS и других ОС.

Flet — это фреймворк на Python, который позволяет создавать веб-приложения, десктопные и мобильные приложения с богатым пользовательским интерфейсом, не имея опыта в веб-разработке. Flet построен на базе Google Flutter, но скрывает всю сложность работы с ним. Программист пишет код на Python, а пользователь видит современное, красивое окно приложения.

Процесс создания приложения на Python с Flet можно разделить на следующие этапы:

1. **Создание проекта:** Создание папки и Python-файла (например, main.py).
2. **Импорт библиотеки:** Подключение Flet в начале файла (import flet as ft).
3. **Создание основной функции:** Определение главной функции main(page: ft.Page), которая будет описывать наше приложение.
4. **Создание интерфейса:** Добавление элементов управления (кнопок, текста, полей ввода) на страницу (page), задание их размеров и свойств.
5. **Написание логики:** Программирование реакций на действия пользователя (события).

6. **Запуск приложения:** Запуск приложения с помощью ft.app(target=main).

Чтобы познакомиться с инструментарием, запустим среду разработки VS Code и создадим первое приложение.

1. Убедитесь, что Python и Flet установлены (команда в терминале: pip install flet).
2. Создайте файл main.py.
3. Напишите простейший код:

```
import flet as ft

def main(page: ft.Page):
    # Заголовок окна
    page.title = "Мое первое приложение на Flet"
    # Добавляем на страницу текст
    page.add(ft.Text("Привет, мир!"))

# Запускаем приложение
ft.app(target=main)
```

При запуске (python main.py) вы увидите окно приложения. Вместо набора окон (форма, инспектор, редактор) в Lazarus, в Python мы работаем в одном окне редактора кода, а результат видим в отдельном окне приложения.

Основные инструменты (аналогия с Lazarus)

В Flet работа строится вокруг одного главного объекта — Page (Страница), которая аналогична главной форме (TForm) в Lazarus.

1. **Редактор кода (VS Code):** Здесь вы пишете программный код на Python. Код подсвечивается, есть автодополнение.

- Ключевые слова (import, def, if) выделяются цветом.
- Строки текста (в кавычках) имеют свой цвет.
- Комментарии начинаются с # и выделяются отдельно.

2. **Объект Page (Страница):** Это и есть окно вашего будущего приложения. Вы управляете им через переменную page в функции main.

- page.title — заголовок окна.
- page.bgcolor — цвет фона.
- page.add() — метод для добавления элементов на страницу.

3. **Библиотека компонентов:** Все готовые элементы (кнопки, поля ввода) хранятся в модуле ft. Вы просто создаете их экземпляры (например, ft.Text(), ft.ElevatedButton()) и настраиваете их свойства.

Вопросы для самоконтроля:

1. Чем Python отличается от Python?
2. Что такое фреймворк Flet?
3. Какой объект в Flet является аналогом главной формы TForm?
4. Как добавить текстовую надпись на страницу в Flet?

Лабораторная работа 5. Реализация простых циклических алгоритмов.

Тема 5. Методы реализации типовых алгоритмов. Операторы и операции.

Цель: изучить работу циклических алгоритмов.

Оборудование: ПК, ОС Windows, MS Office, Python, методические указания по выполнению лабораторного занятия.

Ход работы:

Цикл с параметром

Оператор цикла применяется при выполнении расчетов или других действий, повторяющихся определенное количество раз. Оператор имеет вид:

For i:= N1 To N2 Do "оператор";

либо

For i:= N1 DownTo N2 Do "оператор";

Здесь i - параметр цикла (переменная порядкового типа),

N1, N2 - начальное и конечное значения параметра цикла i.

N1, N2 могут быть константами, переменными или выражениями порядкового типа.

Напомним, что "оператор" может иметь вид: Begin "операторы" end;

В случае связки "To" цикл выполняется при условии N1 <= N2 и происходит с единичным возрастанием параметра цикла i от N1 до N2. В случае связки DownTo цикл выполняется при условии N1 >= N2 и происходит с единичным уменьшением параметра цикла i от N1 до N2.

В операторе цикла не разрешается присваивать параметру цикла какое-либо значение.

После окончания цикла значение параметра цикла "i" неопределенно.

Оператор цикла часто применяется для суммирования значений некоторой последовательности чисел или значений функции при известном числе операций суммирования. Напомним некоторые определения, связанные с расчетом суммы последовательности.

Сумма членов последовательности величин

$a_1, a_2, a_3, \ldots, a_n$

называется конечной суммой

$S_n = a_1 + a_2 + a_3 + \ldots + a_n$

Для некоторых последовательностей известны формулы расчета конечных сумм, например:

при $a_n = a_n\text{-}1 + d$; $S_n = (a_1 + a_n)*n/2$; - арифметическая прогрессия,

при $a_n = a_n\text{-}1 * q$; $S_n = (a_1 - a_n*q)/(1-q)$; - геометрическая прогрессия,

где d и q - постоянные числа.

Здесь N-ый член последовательности выражается через (N-1)-ый член. Такие зависимости называются реккурентными.

Конечная сумма последовательности может быть неизвестна, тогда для ее расчета применяется алгоритм суммирования членов последовательности в цикле от 1 до N. Приведем пример расчета конечной суммы последовательности: $12 + 32 + 52 + \ldots + (2*N\text{-}1)_2$; $S_n = N*(4*N_2\text{-}1)/3$;

Программа расчета конечной суммы

```
print("Введите число членов суммы N=", end='')
N = int(input())

S = 0
for i in range(1, N + 1):    # цикл суммирования
   a = (2 * i - 1) ** 2      # a = (2i-1)^2
   S = S + a

# Расчет по формуле: Sn = N*(4*N*N-1) // 3
Sn = N * (4 * N * N - 1) // 3

print(f"Конечная сумма S={S:10.2f}")
print(f"Расчет конечной суммы по формуле Sn={Sn:10.2f}")
print("Нажмите Enter для выхода...")
input()
```

В некоторых случаях "N"-ый член последовательности определяется через сумму предыдущих членов, например,

$a_n = p*S_{n-1}$,

тогда

$S_n = S_{n-1} + a_n = S_{n-1}*(1+p)$,

и конечную сумму можно рассчитать по формуле:

$S_n = S_0*(1+p)N$,

где "S_0" - начальная сумма.

Рассмотрим программу вычисления конечной суммы денежного вклада в банк через N месяцев при ежемесячной процентной ставке "pr" (5% соответствует pr=5).

```
# Программа расчета конечной суммы вклада в банк

print("Введите начальную сумму вклада S=", end='')
S = float(input())

print("Введите процент по вкладу pr=", end='')
pr = float(input())

print("Введите количество месяцев вклада N=", end='')
N = int(input())

# Расчет через цикл
for i in range(1, N + 1):
   S = S * (1 + pr / 100)

print(f"Конечная сумма вклада S={S:10.2f}")

# Расчет конечной суммы вклада по формуле сложных процентов
Sn = float(input("Введите начальную сумму для расчета по формуле Sn="))
# Формула сложных процентов: Sn = S * (1 + pr/100)^N
```

```
Sn = Sn * (1 + pr / 100) ** N
print(f"Расчет конечной суммы вклада по формуле Sn={Sn:10.2f}")

print("Нажмите Enter для выхода...")
input()
```

Часто применяются вложенные операторы цикла. Например, если необходимо провести все варианты расчета при изменении нескольких параметров в заданных диапазонах.

Составим программу расчета функции y = A*sin(x) - cos(x)/A; при изменении аргумента "x" в диапазоне от 0 до Pi с шагом Pi/100 и при изменении параметра "A" в диапазоне от 1 до 3 с шагом 0.5.

```
import math

print(' Расчет по формуле:  y=A*sin(x)-cos(x)/A; ')
print('-----------------------------------------------')
print('|   X   |  A=1.0 | A=1.5 | A=2.0 | A=2.5 | A=3.0 |')
print('-----------------------------------------------')

dx = math.pi / 100

for i in range(0, 101):  # внешний цикл изменения аргумента "X"
   x = dx * i
   print(f"{x:8.4f}", end='')

   for j in range(1, 6):  # вложенный цикл изменения параметра "A"
      A = 0.5 * (j + 1)
      y = A * math.sin(x) - math.cos(x) / A
      print(f"{y:8.4f}", end='')

   print()  # перевод курсора на новую строчку

   # задержка прокрутки экрана до нажатия Enter
   if ((i + 1) % 20) == 0:
      input("Нажмите Enter для продолжения...")

input("Нажмите Enter для выхода...")
```

Вопросы для самоконтроля

1. Переменные.
2. Типы данных.
3. Объявление и инициализация переменных.
4. Область действия и время существования переменных.
5. Константы.
6. Операторы и операции.
7. Ввод – вывод данных.
8. Операторы присваивания.

Лабораторная работа 6. Реализация алгоритмов обработки одномерных и двумерных массивов.

Тема 5. Методы реализации типовых алгоритмов. Операторы и операции.

Цель: изучить работу одномерных массивов.

Оборудование: ПК, ОС Windows/Linux, Python 3, среда разработки, методические указания.

Ход работы:

Одномерные массивы

Одномерный массив – это именованная последовательность, состоящая из пронумерованных элементов одного типа. Элементы могут быть любого имеющегося в **Python** (за исключение файлового) типа данных. Номер, также называемый индексом, имеет каждый элемент массива. Индекс должен быть порядкового типа. Одномерный массив можно объявить как в качестве переменной:

var <имя переменной>: array[m..n] of <тип элементов>;

так и типа:

type <имя типа> = array[m..n] of <тип элементов>;

Здесь **m** – номер первого элемента, а **n** – последнего. Например, если диапазон задан так: [1..10], то это означает, что определен одномерный массив размерностью в 10 элементов, с индексами от 1 до 10.

Для обращения к элементу массива нужно указать его имя и номер: mas[i], тут mas – имя, i – номер. В программе ниже мы объявим массив и произведем простые операции над его элементами.

```
# Программа для работы с массивом

# Создаем список (массив) из 10 элементов, заполненных нулями
mas = [0.0] * 10
A = [0.0] * 10

# Присваиваем значения элементам
mas[0] = 32    # в Python индексация начинается с 0
mas[4] = 13    # mas[5] в Pascal соответствует mas[4] в Python
mas[8] = 43    # mas[9] в Pascal соответствует mas[8] в Python

# Вычисляем значение
A[0] = (mas[8] - mas[0]) * mas[4]

# Выводим результат с форматированием (5 символов, 2 знака после запятой)
print(f"{A[0]:5.2f}")

# Ожидание нажатия клавиши (аналог readkey)
input("Нажмите Enter для выхода...")
```

В каком-то смысле с массивами можно работать, как и с обычными переменными, но представьте, например ситуацию, когда необходимо заполнить массив, состоящий из десятков или тысяч элементов. Это будет

удобней сделать посредством цикла. Следующая конструкция заполняет массив числами и выводит их на экран.

```
for i in range(1, n + 1):
   mas[i] = i
   print(f"{mas[i]:3}", end='')
```

Если необходимо, чтобы массив состоял из значений, введенных с клавиатуры, то просто замените присвоение на оператор read. Также бывают ситуации, когда требуется заполнить массив случайными числами. Программа ниже поочередно присваивает каждому элементу случайную величину.

```
import random
import os

# Очистка экрана (аналог clrscr)
os.system('cls' if os.name == 'nt' else 'clear')

# Инициализация генератора случайных чисел
random.seed()

# Создаем массив из 100 элементов
mas = [0] * 100

# Заполняем массив случайными числами от 0 до 9 и выводим
for i in range(100):
   mas[i] = random.randint(0, 9)  # random(10) в Pascal дает числа от 0 до 9
   print(f"{mas[i]:2}", end='')

# Ожидание нажатия клавиши (аналог readkey)
input("\n\nНажмите Enter для выхода...")
```

Широко распространены задачи связанные с разного рода алгоритмами применимыми к массивам. Среди них особенно популярны методы поиска и сортировки элементов.

Двумерные массивы

Тем, кто знакомым с математическими матрицами, будет не трудно освоить и **двумерные массивы в Python**. Матрица – это математический объект, представляющий собой прямоугольную таблицу. Таблица состоит из элементов, которые находятся на пересечении строк и столбцов, определяющих их, то есть i-ая строка и j-ый столбец задают адрес k-ому элементу матрицы (k_{ij}). Двумерные массивы абсолютно аналогичны математическим матрицам, поэтому их можно представить так:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \dots & & & \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix}$$

В отличие от одномерных массивов, двумерные характеризуются в программе парой индексов, один из которых соответствует номеру строки, другой – столбца:

Mas[m, n], где Mas – имя массива, n – номер строки, а m – номер столбца.

Описать матрицу в программе можно несколькими способами:

1) В разделе описания переменных:

Var Mas: Array[1..n, 1..m] of <тип элементов>;

2) При помощи одномерного массива, элементами которого являются одномерные массивы.

Пример:

```
# Константы
n = 5
m = 10

# Создание двумерного массива (матрицы) размером n x m
# В Python тип элементов определяется автоматически при заполнении

# Способ 1: Создание пустого двумерного массива
Mas = [[0] * m for _ in range(n)]
# Это создает матрицу 5x10, заполненную нулями

# Способ 2: Создание двумерного массива с явным указанием типа (например, float)
Mas_float = [[0.0] * m for _ in range(n)]

# Способ 3: Создание двумерного массива и заполнение значениями позже
Mas_empty = [[None] * m for _ in range(n)]
```

Переменная Mas – матрица, состоящая из пяти строк, в каждую из которых включено по десять элементов.

В следующей программе массив сначала заполняется числами с клавиатуры, а затем выводиться на экран.

```
# Программа для ввода и вывода двумерного массива

n = 3  # количество строк
m = 3  # количество столбцов

# Создаем пустой двумерный массив (матрицу)
mas = [[0] * m for _ in range(n)]
```

```
# Ввод массива
print("Введите элементы матрицы 3x3:")
for i in range(n):
    for j in range(m):
        mas[i][j] = int(input(f'Элемент {i+1} строки, {j+1} столбца = '))

# Вывод массива
print("\nПолучившаяся матрица:")
for i in range(n):
    for j in range(m):
        print(f"{mas[i][j]:5}", end='')
    print()  # переход на новую строку
```

Стоит отметить, что для вывода двумерного массива в виде таблицы необходимо указать перевод строки после выхода из внутреннего цикла.

Количество элементов в массиве (его размерность) можно узнать, умножив количество строк на количество столбцов.

Вопросы для самоконтроля

1. Операторы отношения.
2. Оператор выбора.
3. Операторы перехода.
4. Операторы цикла.
5. Принудительный выход из цикла.
6. Переменные.
7. Типы данных.
8. Объявление и инициализация переменных.
9. Область действия и время существования переменных.
10. Константы.
11. Операторы и операции.
12. Ввод – вывод данных.
13. Операторы присваивания.

Лабораторная работа 7. Реализация алгоритмов обработки текстовых данных, поиск и ввода/вывода.

Тема 6. Операторы отношения. Операторы цикла.

Цель: изучить алгоритмы работы с текстовыми данными.

Оборудование: ПК, ОС Windows/Linux, Python 3, среда разработки, методические указания.

Ход работы:

Обработка текстовых файлов в языке Free Python

При работе с текстовыми файлами следует учесть следующее:

1. Действие процедур reset, rewrite, close, rename, erase и функции eof аналогично их действию при работе с компонентными (типизированными) файлами.

2.Процедуры seek, truncate и функция filepos не работают с текстовыми файлами.

3.При работе с текстовыми файлами можно пользоваться процедурой append(f), где f – имя файловой переменной, которая служит для специального открытия файлов для дозаписи. Она применима только к уже физически существующим файлам, открывает и готовит их для добавления информации в конец файла.

4. Запись и чтение в текстовый файл осуществляются с помощью процедур write, writeln, read, readln следующей структуры:

read(f, x1, x2, x3, ..., xn); read(f, x);

readln(f, x1, x2, x3, ..., xn); readln(f, x);

write(f, x1, x2, x3, ..., xn); write(f, x);

writeln(f, x1, x2, x3, ..., xn); writeln(f, x);

В этих операторах f — файловая переменная. В операторах чтения (read, readln) x, x1, x2, x3, …, xn — переменные, в которые происходит чтение из файла. В операторах записи write, writeln x, x1, x2, x3, …, xn — переменные или константы, информация из которых записывается в файл.

Есть ряд особенностей при работе операторов write, writeln, read, readln с текстовыми файлами. Имена переменных могут быть целого, вещественного, символьного и строкового типа. Перед записью данных в текстовый файл с помощью процедуры write происходит их преобразование в тип string. Действие оператора writeln отличается тем, что записывает в текстовый файл символ конца строки.

При чтении данных из текстового файла с помощью процедур read, readln происходит преобразование из строкового типа к нужному типу данных. Если преобразование невозможно, то генерируется код ошибки, значение которого можно узнать, обратившись к функции IOResult. Компилятор FreePython позволяет генерировать код программы в двух режимах: с проверкой корректности ввода-вывода и без нее.

В программу может быть включен ключ режима компиляции. Кроме того, предусмотрен перевод контроля ошибок ввода-вывода из одного состояния в другое: {$I+} – режим проверки ошибок ввода-вывода включен; {$I-} – режим проверки ошибок ввода-вывода отключен.

По умолчанию, как правило, действует режим {$I+}. Можно многократно включать и выключать режимы, создавая области с контролем ввода и без него. Все ключи компиляции описаны в приложении.

При включенном режиме проверки ошибка ввода-вывода будет фатальной, программа прервется, выдав номер ошибки.

Если убрать режим проверки, то при возникновении ошибки вво- да-вывода программа не будет останавливаться, а продолжит работу со следующего оператора. Результат операции ввода-вывода будет не определен.

Для опроса кода ошибки лучше пользоваться специальной функцией IOResult, но необходимо помнить, что опросить ее можно только один раз после каждой операции ввода или вывода: она обнуляет свое значение при каждом вызове. IOResult возвращает целое число, соответствующее коду последней ошибки ввода-вывода. Если IOResult=0, то при вводе-выводе ошибок не было, иначе IOResult возвращает код ошибки. Некоторые коды ошибок приведены в таблице.

Таблица. Коды ошибок

Код ошибки	**Описание**
2	файл не найден
3	путь не найден
4	слишком много открытых файлов
5	отказано в доступе
12	неверный режим доступа
15	неправильный номер диска
16	нельзя удалять текущую директорию
100	ошибка при чтении с диска
101	ошибка при записи на диск
102	не применена процедура Assign
103	файл не открыт
104	файл не открыт для ввода
105	файл не открыт для вывода
106	неверный номер
150	диск защищён от записи

Рассмотрим несколько практических примеров обработки ошибок ввода-вывода:

1. При открытии проверить, существует ли заданный файл и возможно ли чтение данных из него.

```
assign (f, 'abc.dat');
{$I-} reset(f); {$I+}
if IOResult<>0 then
writeln ('файл не найден или не читается') else
```

```
begin read(f,...);
…
close(f);
end;
```

2. Проверить, является ли вводимое с клавиатуры число целым. var i:integer;

```
begin {$I-} repeat
write('введите целое число i'); readln(i);
until (IOResult=0); {$I+}
```

{ Этот цикл повторяется до тех пор, пока не будет введено целое число} end.

При работе с текстовым файлом необходимо помнить специальные правила чтения значений переменных:

•если вводятся числовые значения, то два числа считаются разделенными, если между ними есть хотя бы один пробел, или символ табуляции, или символ конца строки;

•при вводе строк начало текущей строки идет сразу за последним введенным до этого символом. Вводится количество символов, равное объявленной длине строки. Если при чтении встретился символ «конец строки», то работа с этой строкой заканчивается. Сам символ конца строки является разделителем и в переменную никогда не считывается;

•процедура readln считывает значения текущей строки файла, курсор переводится в новую строку файла, и дальнейший ввод осуществляется с нее.

Считать из файла *input.txt* числа (числа записаны в столбик). Затем записать их сумму в файл *output.txt*

```
var p, x: integer;
f: text;
begin
    assign(f, 'input.txt');
    reset(f);
    p := 1;
    while not eof(f) do begin
        readln(f, x);
        p := p + x;
    end;
    close(f);
    assign(f, 'output.txt');
    rewrite(f);
    writeln(f, 'Сумма чисел ', p);
    close(f);
end.
```

Считать из файла *input.txt* числа (числа записаны в столбик). Затем записать их произведение в файл *output.txt*

```
var p, x: integer;
f: text;
begin
```

```
	assign(f, 'input.txt');
	reset(f);
	p := 1;
	while not eof(f) do begin
		readln(f, x);
		p := p * x;
	end;
	close(f);
	assign(f, 'output.txt');
	rewrite(f);
	writeln(f, 'Произведение чисел ', p);
	close(f);
end.
```

Логический тип данных.

До сих пор мы писали линейные программы, то есть сначала выполняли первое действие, потом второе и т.д. Сегодня мы хотим поговорить с Вами о логическом типе данных.

Скажем у нас есть такая запись - x-y>10

Из этой следует условие, то есть если из x вычесть y то будет ли эта разность больше десяти. В этой записи нам нужно знать x и y - допустим x=11, а y=0, то x-y>10 -> 11-0>10 получается что условие верно - разность 11 и 0 больше 10. Об этом выражении можно сказать что оно булевое или логическое.

Название булевое произошло от имени Джорджа Буля - разработчика булевой логики. Переменная, которая может принимать одно из двух значений - true(истина) и false(ложь) - называется логической или булевой, но мы её будем называть так, как принято - логическая.

Для начала давайте рассмотрим пример простой программы:

```
Program logika;
uses crt;
var a, b: Boolean;
begin
	clrscr;
	a := true;
	b := false;
	write(a, ' ', b);
	readln;
end.
```

В этой программе мы объявили две переменные логического типа - логический тип в Python обозначается так - Boolean.

Потом переменным присвоили логические значения - true(истина) и false(ложь), после чего вывели их через пробел. Попробуйте скопировать к себе код этой программы и скомпилировав запустить его - на экране появятся две записи через пробел - TRUE FALSE.

Попробуем немного изменить код программы - пусть переменная a будет результатом сравнения двух чисел:

```
Program logika;
uses crt;
var a: Boolean;
begin
        clrscr;
        a := 2>4;
        write(a);
        readln;
end.
```

Мы проверили - больше ли двойка чем четыре, и после выполнения программы получим результат - FALSE - что значит ложь. Также можно проверить любые другие числа, например - 10 и 5, тогда результат будет TRUE, что значит истина. Также можно сравнивать и слова, но это пока нам не нужно.

Для последнего примера мы Вам предоставим код программы, которая считывает два числа с клавиатуры и проверяет их на все возможные сравнения - то есть больше, меньше, равно, и т.д.

И выводит на экран ряд сообщений типа - TRUE или FALSE и к ним приписывает какое сравнение было произведено:

```
Program logika;
uses crt;
var a: Boolean;
   num1, num2: Integer;
begin
        clrscr;
        write('Введите через пробел два числа - ');
        readln(num1, num2);
        a := num1 = num2;
        writeln('Первое число равно второму - ', a);
        a := num1 > num2;
        writeln('Первое число больше второго - ', a);
        a := num1 < num2;
        writeln('Первое число меньше второго - ', a);
        a := num1 <> num2;
        write('Первое число не равно второму - ', a);
        readln;
end.
```

Ввод-вывод вещественных чисел.

В Python вещественное число представляется определённым образом, а именно с помощью десятичного признака и указания степени. Давайте создадим новую программу и в ней пропишем три переменных вещественного типа, где две переменных будем считывать с клавиатуры, и третей переменной присвоим произведение двух вещественных чисел, считанных с клавиатуры:

```
Program Real_Num;
uses crt;
var num1, num2, res: Real;
```

```
begin
	clrscr;
	write('Введите два вещественных числа через пробел - ');
	readln(num1, num2);
	res := num1 + num2;
	write('Сумма двух вещественных чисел - ', res);
	readln;
end.
```

Теперь выполним нашу программу и попробуем ввести два вещественных числа, например - 2.3212 5.2313 - и не стоит забывать что числа нужно вводить с точкой, а не с запятой.

И после получим результат - 1.21428935600000E+001. До знака плюс идёт десятичный признак, а после степень. Но такая запись нам не удобна. Для изменения вида в Python используется специальная запись:

```
Program Real_Num;
uses crt;
var num1, num2, res: Real;
begin
	clrscr;
	write('Введите два вещественных числа через пробел - ');
	readln(num1, num2);
	res := num1 * num2;
	write('Произведение двух вещественных чисел - ', res:4:4);
	readln;
end.
```

В выводе переменной с произведением мы внесли некоторые изменения, а именно - после переменной res поставили двоеточие и кол-во знаков до запятой - у нас их четыре, потом поставили ещё двоеточие и кол-во знаков после запятой - у нас их столько же. Вот такая запись используется в Python.

Если знаков до запятой меньше чем мы указали, то при выводе результата он у нас сместиться на недостающее кол-во цифр в право, то есть до видимого результата будут стоять невидимые нули, которые будут сдвигать ответ вправо.

Вопросы для самоконтроля

1. Массивы: определение, виды.
2. Объявление одномерного массива.
3. Ввод и вывод одномерных массивов.
4. Обработка одномерных и двумерных массивов.
5. Управляющие структуры.
6. Понятие потока.
7. Механизм буферизации.
8. Понятие подпрограммы.
9. Библиотеки среды разработки
10. Операторы отношения.
11. Оператор выбора.
12. Операторы перехода.

13. Операторы цикла.
14. Принудительный выход из цикла.

Лабораторная работа 8. Создание простейших классов в Python (tkinter)

Тема: Методы, инкапсуляция, свойства класса, наследование.

Цель: Изучить основы создания GUI-приложений на Python с использованием библиотеки tkinter, научиться работать с типами данных, функциями преобразования и обработкой событий.

Оборудование: ПК, ОС Windows/Linux, Python 3, библиотека tkinter, среда разработки (IDLE, PyCharm, VS Code), методические указания.

Ход работы

1) Типы данных в Python

В Python все данные — это объекты. Они хранятся в памяти и могут быть различных типов: целые числа (int), числа с плавающей точкой (float), строки (str), списки (list) и другие.

Типы данных определяют, какие операции можно выполнять с объектами и как они хранятся в памяти.

Переменная — это ссылка на объект в памяти. У переменной есть имя и значение. Тип переменной определяется автоматически при присваивании значения.

Для объявления переменной не нужно использовать специальное слово. Достаточно просто присвоить значение.

Примеры:

```
a = 10          # целочисленная переменная (int)
b = 3.14        # вещественная переменная (float)
c = "Привет"    # строковая переменная (str)
```

Целочисленный тип данных (int)

В Python целые числа не ограничены фиксированным размером (зависит от доступной памяти). Для удобства можно привести диапазоны для типов, аналогичных Free Pascal:

Тип в Python	Аналог в Pascal	Диапазон (примерно)	Размер (байт)
int	Shortint, Integer, Longint	Не ограничен	зависит от значения

Вещественный тип данных (float**)**

Внутреннее представление вещественных чисел соответствует стандарту IEEE 754 (как Double в Pascal).

Тип в Python	Аналог в Pascal	Диапазон (примерно)	Точность (значащих цифр)	Размер (байт)
float	Double	2.3e-308 .. 1.7e308	15-16	8

Операции и выражения

Выражение задает порядок выполнения действий над данными.

Оператор	Действие
+	Сложение
-	Вычитание
*	Умножение
/	Деление (возвращает float)
//	Целочисленное деление
%	Остаток от деления
**	Возведение в степень

Примеры:

```
x = 15 // 7   # результат: 2
y = 15 % 7    # результат: 1
z = 2 ** 3    # результат: 8
```

Стандартные функции

Python предоставляет множество встроенных функций. Для математических операций нужно импортировать модуль math.

Функция	Действие
abs(n)	Абсолютное значение n.
math.sqrt(n)	Квадратный корень из n.
pow(n, 2) или n ** 2	Квадрат n.
math.exp(n)	Экспонента n (e в степени n).
math.log(n)	Натуральный логарифм n.
random.randint(0, n-1)	Случайное целое в диапазоне.
random.random()	Случайное число от 0 до 1.
math.sin(x)	Синус угла x в радианах.
math.cos(x)	Косинус угла x в радианах.
math.atan(x)	Арктангенс x.

Для преобразования градусов в радианы: radians = degrees * math.pi / 180.

Функции преобразования типов

В Python преобразование типов выполняется с помощью функций с именами типов.

Функция	Действие
chr(n)	Символ по коду n.
str(n)	Преобразует число n в строку.
int(s)	Преобразует строку s в целое число.
float(s)	Преобразует строку s в вещественное число.
round(n, k)	Округляет число n до k знаков после запятой.
int(n)	Отбрасывает дробную часть вещественного числа.
n % 1	Получает дробную часть числа.

Проект «Калькулятор» на Python (tkinter)

Задание: Создать приложение для вычисления суммы двух чисел. Окно программы должно содержать три поля ввода, кнопку и метки.

Ход выполнения

1. Создайте новый файл calculator.py.
2. Напишите код, реализующий интерфейс.

```
import tkinter as tk
from tkinter import messagebox

def calculate_sum():
    try:
        # Получаем текст из полей ввода и преобразуем в числа
        num1 = float(entry1.get())
```

```
            num2 = float(entry2.get())
            result = num1 + num2
            # Выводим результат в третье поле
            entry3.delete(0, tk.END)
            entry3.insert(0, str(result))
        except ValueError:
            messagebox.showerror("Ошибка", "Введите корректные числа!")

    # Создаем главное окно
    root = tk.Tk()
    root.title("Калькулятор")
    root.geometry("300x200")

    # Метки и поля ввода
    label1 = tk.Label(root, text="Первое число:")
    label1.pack(pady=5)

    entry1 = tk.Entry(root)
    entry1.pack(pady=5)

    label2 = tk.Label(root, text="Второе число:")
    label2.pack(pady=5)

    entry2 = tk.Entry(root)
    entry2.pack(pady=5)

    label3 = tk.Label(root, text="Результат:")
    label3.pack(pady=5)

    entry3 = tk.Entry(root)
    entry3.pack(pady=5)

    # Кнопка вычисления
    button = tk.Button(root, text="Вычислить", command=calculate_sum)
    button.pack(pady=10)

    # Запуск приложения
    root.mainloop()
```

3. Сохраните файл и запустите его. Проверьте работу приложения.

2) Организация ввода и вывода данных

В Python (tkinter) для ввода и вывода данных можно использовать:

- Поля ввода (Entry) — аналог Edit в Lazarus.
- Метки (Label) — аналог Label для вывода.

- Диалоговые окна — simpledialog.askstring, messagebox.showinfo и т.д.

Ввод данных с помощью диалогового окна

Для этого используется модуль tkinter.simpledialog.

```
from tkinter import simpledialog

# Функция для ввода строки
name = simpledialog.askstring("Ввод данных", "Введите ваше имя:")

# Для ввода числа
age = simpledialog.askinteger("Ввод данных", "Введите ваш возраст:")

# Для ввода вещественного числа
weight = simpledialog.askfloat("Ввод данных", "Введите ваш вес:")
```

Вывод данных

Вывод в окне сообщения (аналог ShowMessage):

```
from tkinter import messagebox

messagebox.showinfo("Результат", f"Вес в килограммах: {kg}")
```

Вывод в метку (Label):

```
result_label.config(text=f"{kg} кг")
```

Практическая работа

Задание: Создать приложение с двумя кнопками: "Ввод данных" и "Выход". Пользователь должен ввести фамилию, имя и возраст с помощью диалоговых окон. По окончании ввода данные должны отобразиться в метке на форме.

Ход выполнения

1. Создайте новый файл anketa.py.
2. Напишите код приложения.

```
import tkinter as tk
from tkinter import simpledialog, messagebox

def enter_data():
    # Ввод данных через диалоговые окна
    surname = simpledialog.askstring("Ввод данных", "Введите фамилию:")
    if surname is None: return  # если нажата отмена
```

```
    name = simpledialog.askstring("Ввод данных", "Введите имя:")
    if name is None: return

    age = simpledialog.askinteger("Ввод данных", "Введите возраст:")
    if age is None: return

    # Формируем строку и выводим в метку
    info = f"{surname} {name}, возраст: {age}"
    result_label.config(text=info)

def exit_app():
    root.destroy()

# Создание окна
root = tk.Tk()
root.title("Анкетные данные")
root.geometry("400x200")

# Кнопки
btn_enter = tk.Button(root, text="Ввод данных", command=enter_data)
btn_enter.pack(pady=10)

btn_exit = tk.Button(root, text="Выход", command=exit_app)
btn_exit.pack(pady=10)

# Метка для вывода результата
result_label = tk.Label(root, text="", font=("Arial", 12))
result_label.pack(pady=20)

root.mainloop()
```

3. Сохраните файл и запустите. Проверьте работу приложения.

Контрольные вопросы для самопроверки

1. Что такое метод в классе Python? Как его объявить и вызвать?
2. Инкапсуляция в Python: как реализовать сокрытие данных? (Приватные атрибуты, свойства @property).
3. Свойства класса: для чего нужны декораторы @property, @setter, @deleter? Приведите пример.
4. В чем отличие // от / в Python?
5. Как преобразовать строку в число с плавающей точкой?

Лабораторная работа 9. Создание классов, иерархически связанных между собой

Тема: Иерархия классов. Интерфейсы.

Цель: Изучить основы создания приложений с несколькими формами в Python (tkinter), научиться работать с переключателями (Radiobutton), циклами и обработкой событий между формами.

Оборудование: ПК, ОС Windows/Linux, Python 3, библиотека tkinter, среда разработки.

Ход работы

1) Компонент Radiobutton (аналог TRadioGroup). Приложение с несколькими формами

Добавление нового окна (формы) в tkinter

В tkinter каждое окно создается как отдельный класс, наследующий от tk.Toplevel или tk.Tk. Главное окно обычно создается от tk.Tk, а дополнительные окна — от tk.Toplevel.

Для создания новой формы нужно определить новый класс и создать его экземпляр при необходимости.

Для показа окон используются методы:

- show() — показывает окно (но в tkinter окно создается и показывается при вызове конструктора).
- grab_set() — делает окно модальным (блокирует другие окна).
- wait_window() — ожидает закрытия окна.

Различие между обычным и модальным окном:

- **Обычное окно** — можно переключаться между окнами.
- **Модальное окно** — блокирует работу с родительским окном до закрытия.

Пример создания модального окна:

```
def show_modal():
    modal = tk.Toplevel(root)
    modal.title("Модальное окно")
    modal.transient(root)  # привязываем к родителю
    modal.grab_set()       # делаем модальным
    root.wait_window(modal)  # ожидаем закрытия
```

Проект «Три формы»

Задание: Создать приложение с тремя формами: Главная, Опции и О программе. Форму Опции вызывать в обычном окне. Для вызова формы О программе использовать модальное окно.

Ход выполнения проекта

1. Создайте новый файл main.py.
2. Реализуйте главное окно (MainWindow), окно настроек (OptionsWindow) и окно "О программе" (AboutWindow).

```
import tkinter as tk
from tkinter import messagebox

class OptionsWindow(tk.Toplevel):
    """Окно настроек (обычное)"""
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.title("Опции")
        self.geometry("300x250")

        # Группа переключателей для выбора цвета
        self.color_var = tk.StringVar(value="white")

        tk.Label(self, text="Цвет главной формы:").pack(pady=5)

        colors = [("Белый", "white"), ("Красный", "red"),
                  ("Синий", "blue"), ("Зеленый", "green")]

        for text, value in colors:
            rb = tk.Radiobutton(self, text=text, variable=self.color_var,
                                value=value)
            rb.pack(anchor='w', padx=20)

        # Заголовок главного окна
        tk.Label(self, text="Заголовок главного окна:").pack(pady=5)
        self.title_entry = tk.Entry(self)
        self.title_entry.insert(0, parent.title())
        self.title_entry.pack(pady=5)

        # Кнопки
```

```
        btn_frame = tk.Frame(self)
        btn_frame.pack(pady=10)

        tk.Button(btn_frame, text="OK",
command=self.apply_settings).pack(side='left', padx=5)
        tk.Button(btn_frame, text="Выход", command=self.destroy).pack(side='left',
padx=5)

    def apply_settings(self):
        """Применение настроек"""
        # Изменяем цвет главной формы
        self.parent.configure(bg=self.color_var.get())
        # Изменяем заголовок
        self.parent.title(self.title_entry.get())
        # Закрываем окно
        self.destroy()

class AboutWindow(tk.Toplevel):
    """Окно 'О программе' (модальное)"""
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.title("О программе")
        self.geometry("250x150")

        # Делаем окно модальным
        self.transient(parent)
        self.grab_set()

        # Содержимое
        tk.Label(self, text="Программа 'Три формы'\nВерсия 1.0\n\nПример
работы с несколькими окнами").pack(pady=20)
        tk.Button(self, text="Выход", command=self.destroy).pack()

class MainWindow(tk.Tk):
    """Главное окно"""
    def __init__(self):
        super().__init__()
        self.title("Главная")
        self.geometry("300x150")

        # Кнопки
        tk.Button(self, text="Опции", command=self.show_options).pack(pady=5)
        tk.Button(self, text="О программе",
command=self.show_about).pack(pady=5)
        tk.Button(self, text="Закрыть", command=self.destroy).pack(pady=5)
```

```
    def show_options(self):
        """Показать окно настроек (обычное)"""
        OptionsWindow(self)

    def show_about(self):
        """Показать окно 'О программе' (модальное)"""
        AboutWindow(self)

# Запуск приложения
if __name__ == "__main__":
    app = MainWindow()
    app.mainloop()
```

3. Сохраните файл и запустите. Проверьте работу:

- Кнопка "Опции" открывает обычное окно (можно переключаться)
- Кнопка "О программе" открывает модальное окно (блокирует главное)
- В окне настроек можно изменить цвет и заголовок главного окна

2) Операторы повторения (циклы) в Python

В Python циклы реализуются с помощью конструкций for и while.

Цикл for ... in

Используется, когда количество повторений известно заранее.

Общий вид:

```
for счетчик in range(нач_знач, кон_знач + 1):
    # тело цикла
```

range(нач_знач, кон_знач) генерирует последовательность от нач_знач до кон_знач - 1.

Пример таблицы квадратов:

```
tabl = ""
for i in range(1, 6):  # от 1 до 5 включительно
    tabl += f"{i}  {i*i}\n"
```

Если нужен обратный порядок: range(кон_знач, нач_знач - 1, -1)

Цикл с условием while

Используется, когда число повторений заранее неизвестно.

Общий вид:

python

```
while условие:
    # тело цикла
```

Тело выполняется, пока условие истинно (**True).**

Цикл с постусловием (аналог repeat ... until)

В Python нет прямого аналога repeat ... until, но можно использовать while True с break:

```
while True:
    # тело цикла
    if условие_выхода:
        break
```

Практическая работа

Задание: Напишите программу, определяющую доход по вкладу с учетом выбранных простых или сложных процентов. Простые проценты начисляются по окончании срока вклада, сложные проценты начисляются ежемесячно и прибавляются к сумме вклада.

Ход выполнения работы

1. Создайте новый файл deposit.py.
2. Реализуйте интерфейс и логику расчета.
   python

```
import tkinter as tk
from tkinter import messagebox

class DepositCalculator(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Доход по вкладу")
        self.geometry("350x250")

        # Переменная для типа процентов
        self.interest_type = tk.StringVar(value="simple")

        self.create_widgets()

    def create_widgets(self):
        # Поля ввода
        tk.Label(self, text="Сумма вклада:").grid(row=0, column=0, padx=5, pady=5, sticky='e')
        self.sum_entry = tk.Entry(self)
        self.sum_entry.grid(row=0, column=1, padx=5, pady=5)
```

```
        tk.Label(self, text="Процентная ставка (% годовых):").grid(row=1, column=0, padx=5, pady=5, sticky='e')
        self.rate_entry = tk.Entry(self)
        self.rate_entry.grid(row=1, column=1, padx=5, pady=5)

        tk.Label(self, text="Срок вклада (мес.):").grid(row=2, column=0, padx=5, pady=5, sticky='e')
        self.period_entry = tk.Entry(self)
        self.period_entry.grid(row=2, column=1, padx=5, pady=5)

        # Группа переключателей
        tk.Label(self, text="Тип процентов:").grid(row=3, column=0, padx=5, pady=5, sticky='e')

        frame = tk.Frame(self)
        frame.grid(row=3, column=1, padx=5, pady=5, sticky='w')

        tk.Radiobutton(frame, text="Простые", variable=self.interest_type,
                       value="simple").pack(anchor='w')
        tk.Radiobutton(frame, text="Сложные", variable=self.interest_type,
                       value="compound").pack(anchor='w')

        # Кнопка вычисления
        tk.Button(self, text="Вычислить", command=self.calculate).grid(row=4, column=0, columnspan=2, pady=10)

        # Поле для результата
        self.result_label = tk.Label(self, text="", font=("Arial", 10), justify='left')
        self.result_label.grid(row=5, column=0, columnspan=2, pady=5)

    def calculate(self):
        try:
            # Получаем данные
            sum_deposit = float(self.sum_entry.get())
            rate = float(self.rate_entry.get())
            period = int(self.period_entry.get())

            monthly_rate = rate / 100 / 12  # месячная ставка

            if self.interest_type.get() == "simple":
                # Простые проценты
                profit = sum_deposit * monthly_rate * period
                total = sum_deposit + profit
            else:
                # Сложные проценты
                total = sum_deposit
```

```
                for month in range(period):
                    total += total * monthly_rate
                profit = total - sum_deposit

            # Выводим результат
            result_text = f"Доход: {profit:.2f}\nСумма в конце срока: {total:.2f}"
            self.result_label.config(text=result_text)

        except ValueError:
            messagebox.showerror("Ошибка", "Введите корректные числовые значения!")

if __name__ == "__main__":
    app = DepositCalculator()
    app.mainloop()
```

3. Проверьте работу приложения:

- Введите сумму, ставку и срок
- Выберите тип процентов
- Нажмите "Вычислить" и сравните результаты

Контрольные вопросы для самопроверки

1. Что такое иерархия классов в Python? Как организовать наследование?
2. Как создать несколько окон в tkinter и управлять ими?
3. В чем разница между обычным и модальным окном? Как реализовать модальное окно в tkinter?
4. Какие циклы существуют в Python? В чем их отличие?
5. Как работает оператор range() в цикле for?
6. Что такое интерфейсы в Python и как они реализуются (абстрактные классы)?

Лабораторная работа 10. Создание классов для обработки массива данных

Тема: Модификаторы доступа к элементам класса. Методы классов.

Цель: Изучить основы создания приложений с меню и стандартными диалогами в Python (tkinter), научиться работать с файловыми операциями и текстовыми редакторами.

Оборудование: ПК, ОС Windows/Linux, Python 3, библиотека tkinter, среда разработки.

Ход работы

1) Компонент Меню (**Menu**) в tkinter

В tkinter для создания главного меню используется класс Menu. Меню может содержать пункты, подменю, разделители и привязываться к главному окну.

Чтобы добавить меню в приложение:

1. Создайте объект Menu для главного окна.
2. Добавьте пункты меню с помощью метода add_command().
3. Привяжите меню к окну методом config(menu=...).

Пример создания простого меню:

```
menubar = Menu(root)
root.config(menu=menubar)

# Создаем пункт меню "Файл"
file_menu = Menu(menubar, tearoff=0)
menubar.add_cascade(label="Файл", menu=file_menu)

# Добавляем команды в меню "Файл"
file_menu.add_command(label="Открыть", command=open_file)
file_menu.add_command(label="Сохранить", command=save_file)
file_menu.add_separator()
file_menu.add_command(label="Выход", command=root.quit)
```

Практическая работа: Калькулятор с меню

Задание: Добавить главное меню в приложение "Калькулятор". В меню включить арифметические действия: «Сложить» и «Разделить».

https://media/image3.jpeg

Ход выполнения проекта

1. Создайте новый файл calculator_menu.py или откройте существующий проект калькулятора.
2. Реализуйте класс калькулятора с меню:

```
import tkinter as tk
from tkinter import messagebox

class CalculatorWithMenu(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Калькулятор с меню")
        self.geometry("300x200")

        self.create_widgets()
        self.create_menu()

    def create_widgets(self):
        """Создание элементов интерфейса"""
        # Поля ввода
        tk.Label(self, text="Первое число:").pack(pady=5)
        self.num1_entry = tk.Entry(self)
        self.num1_entry.pack(pady=5)

        tk.Label(self, text="Второе число:").pack(pady=5)
        self.num2_entry = tk.Entry(self)
        self.num2_entry.pack(pady=5)

        tk.Label(self, text="Результат:").pack(pady=5)
        self.result_entry = tk.Entry(self)
        self.result_entry.pack(pady=5)

    def create_menu(self):
        """Создание главного меню"""
        menubar = Menu(self)
        self.config(menu=menubar)

        # Меню "Действия"
        action_menu = Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Действия", menu=action_menu)
```

```
        action_menu.add_command(label="Сложить", command=lambda:
    self.calculate('+'))
        action_menu.add_command(label="Разделить", command=lambda:
    self.calculate('/'))
        action_menu.add_separator()
        action_menu.add_command(label="Выход", command=self.quit)

    def calculate(self, operation):
        """Выполнение арифметической операции"""
        try:
            num1 = float(self.num1_entry.get())
            num2 = float(self.num2_entry.get())

            if operation == '+':
                result = num1 + num2
                self.result_entry.delete(0, tk.END)
                self.result_entry.insert(0, str(result))

            elif operation == '/':
                if num2 == 0:
                    messagebox.showerror("Ошибка", "Делить на 0 нельзя!")
                    self.num2_entry.delete(0, tk.END)
                    self.num2_entry.focus()
                    self.result_entry.delete(0, tk.END)
                else:
                    result = num1 / num2
                    self.result_entry.delete(0, tk.END)
                    self.result_entry.insert(0, str(result))

        except ValueError:
            messagebox.showerror("Ошибка", "Введите корректные числа!")

if __name__ == "__main__":
    app = CalculatorWithMenu()
    app.mainloop()
```

3. Проверьте работу приложения:

- Введите числа в поля
- Выберите действие в меню
- Проверьте результат

2) Стандартные диалоги. Создание текстового редактора

В tkinter есть встроенные диалоговые окна для работы с файлами и шрифтами:

- filedialog.askopenfilename() — открытие файла
- filedialog.asksaveasfilename() — сохранение файла
- font.Font и диалог выбора шрифта (tkinter.font)

Для активации диалога нужно вызвать соответствующую функцию, которая возвращает выбранное имя файла или None при отмене.

Проблема кодировок

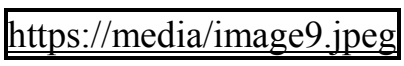
В Windows текстовые файлы часто сохраняются в кодировке CP1251 (ANSI), а Python работает с Unicode (UTF-8). При работе с русским текстом нужно преобразовывать кодировки:

```
# Чтение файла в ANSI и преобразование в UTF-8
with open(filename, 'r', encoding='cp1251') as f:
    text = f.read()

# Запись файла в ANSI
with open(filename, 'w', encoding='cp1251') as f:
    f.write(text)
```

Практическая работа: Текстовый редактор

Задание: Создать приложение "Текстовый редактор" с меню, содержащим пункты: Файл → Открыть, Файл → Сохранить, Шрифт. Приложение должно открывать текстовый файл, отображать его в многострочном поле (Text), позволять редактировать и сохранять. Для выбора файла использовать диалоги, для выбора шрифта — диалог выбора шрифта.



Ход выполнения проекта

1. Создайте новый файл text_editor.py.
2. Реализуйте класс текстового редактора:

```
import tkinter as tk
from tkinter import filedialog, messagebox, font
import os

class TextEditor(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Текстовый редактор")
```

```
        self.geometry("600x400")

        self.current_file = None
        self.create_menu()
        self.create_widgets()

    def create_menu(self):
        """Создание главного меню"""
        menubar = tk.Menu(self)
        self.config(menu=menubar)

        # Меню "Файл"
        file_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Файл", menu=file_menu)
        file_menu.add_command(label="Открыть", command=self.open_file)
        file_menu.add_command(label="Сохранить", command=self.save_file)
        file_menu.add_command(label="Сохранить как...",
command=self.save_as_file)
        file_menu.add_separator()
        file_menu.add_command(label="Выход", command=self.quit)

        # Меню "Шрифт"
        font_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Шрифт", menu=font_menu)
        font_menu.add_command(label="Выбрать шрифт...",
command=self.choose_font)

    def create_widgets(self):
        """Создание текстового поля"""
        # Создаем текстовое поле с прокруткой
        text_frame = tk.Frame(self)
        text_frame.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.text_area = tk.Text(text_frame, wrap=tk.WORD)
        scrollbar = tk.Scrollbar(text_frame, command=self.text_area.yview)
        self.text_area.config(yscrollcommand=scrollbar.set)

        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.text_area.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

        # Начальный шрифт
        self.current_font = font.Font(family="Arial", size=12)
        self.text_area.config(font=self.current_font)

    def open_file(self):
        """Открытие файла"""
        filename = filedialog.askopenfilename(
```

```
            title="Открыть файл",
            filetypes=[("Текстовые файлы", "*.txt"), ("Все файлы", "*.*")]
        )

        if filename:
            try:
                # Пробуем разные кодировки
                encodings = ['utf-8', 'cp1251', 'cp866']
                content = None

                for enc in encodings:
                    try:
                        with open(filename, 'r', encoding=enc) as f:
                            content = f.read()
                        break
                    except UnicodeDecodeError:
                        continue

                if content is None:
                    messagebox.showerror("Ошибка", "Не удалось определить
кодировку файла")
                    return

                self.text_area.delete(1.0, tk.END)
                self.text_area.insert(1.0, content)
                self.current_file = filename
                self.title(f"Текстовый редактор - {os.path.basename(filename)}")

            except Exception as e:
                messagebox.showerror("Ошибка", f"Не удалось открыть файл: {e}")

    def save_file(self):
        """Сохранение файла"""
        if self.current_file:
            self._save_to_file(self.current_file)
        else:
            self.save_as_file()

    def save_as_file(self):
        """Сохранение файла с выбором имени"""
        filename = filedialog.asksaveasfilename(
            title="Сохранить файл",
            defaultextension=".txt",
            filetypes=[("Текстовые файлы", "*.txt"), ("Все файлы", "*.*")]
        )

        if filename:
```

```
            self._save_to_file(filename)
            self.current_file = filename
            self.title(f"Текстовый редактор - {os.path.basename(filename)}")

    def _save_to_file(self, filename):
        """Внутренний метод сохранения в файл"""
        try:
            content = self.text_area.get(1.0, tk.END)
            # Сохраняем в ANSI (cp1251) для совместимости с Windows
            with open(filename, 'w', encoding='cp1251') as f:
                f.write(content)
            messagebox.showinfo("Успех", "Файл успешно сохранен")
        except Exception as e:
            messagebox.showerror("Ошибка", f"Не удалось сохранить файл: {e}")

    def choose_font(self):
        """Выбор шрифта"""
        # Создаем диалог выбора шрифта
        font_dialog = tk.Toplevel(self)
        font_dialog.title("Выбор шрифта")
        font_dialog.geometry("300x200")
        font_dialog.transient(self)
        font_dialog.grab_set()

        # Список доступных шрифтов
        fonts = list(font.families())
        fonts.sort()

        tk.Label(font_dialog, text="Выберите шрифт:").pack(pady=5)

        # Список для выбора шрифта
        listbox = tk.Listbox(font_dialog, height=10)
        listbox.pack(padx=10, pady=5, fill=tk.BOTH, expand=True)

        # Добавляем прокрутку
        scrollbar = tk.Scrollbar(listbox)
        scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        listbox.config(yscrollcommand=scrollbar.set)
        scrollbar.config(command=listbox.yview)

        # Заполняем список
        for f in fonts[:50]:  # Показываем только первые 50 для скорости
            listbox.insert(tk.END, f)

        # Размер шрифта
        tk.Label(font_dialog, text="Размер:").pack(pady=5)
        size_var = tk.IntVar(value=12)
```

```
        size_spinbox = tk.Spinbox(font_dialog, from_=8, to=72,
    textvariable=size_var)
        size_spinbox.pack(pady=5)

        def apply_font():
            selected = listbox.curselection()
            if selected:
                font_name = listbox.get(selected[0])
                font_size = size_var.get()
                self.current_font.config(family=font_name, size=font_size)
                font_dialog.destroy()

        tk.Button(font_dialog, text="Применить",
    command=apply_font).pack(pady=10)

    if __name__ == "__main__":
        app = TextEditor()
        app.mainloop()
```

3. Сохраните и запустите приложение.
4. Проверьте работу:

- Откройте текстовый файл (можно создать любой .txt файл)
- Отредактируйте текст
- Сохраните файл
- Измените шрифт

5. Особенности работы с кодировками:

- Программа пытается открыть файл в разных кодировках (UTF-8, CP1251, CP866)
- При сохранении используется CP1251 для совместимости с Windows Notepad

Дополнительные задания (для углубленного изучения)

1. **Поиск и замена:**
   Добавьте в меню пункт "Правка" с командами "Найти" и "Заменить".
2. **Статистика:**
   Добавьте пункт меню "Статистика", который показывает количество символов, слов и строк в документе.
3. **Недавние файлы:**
   Реализуйте список последних открытых файлов в меню "Файл".
4. **Подсветка синтаксиса:**
   Добавьте простую подсветку синтаксиса для Python-файлов.
5. **Вкладки:**
   Реализуйте интерфейс с вкладками для работы с несколькими файлами одновременно.

Контрольные вопросы для самопроверки

1. Как создать главное меню в tkinter? Как добавить подменю?
2. Какие типы диалоговых окон существуют в tkinter для работы с файлами?
3. Как обработать отмену выбора в диалоговом окне?
4. Какие проблемы могут возникнуть при работе с русским текстом в файлах? Как их решить?
5. Что такое кодировка? Чем отличается UTF-8 от CP1251?
6. Как изменить шрифт текстового поля в tkinter программно?
7. В чем разница между методами add_command и add_cascade при создании меню?
8. Как реализовать модальное окно для выбора шрифта?

Лабораторная работа 11. Создание классов и графических приложений на Python

Тема: Обработка событий, графические методы, классы и виджеты в tkinter.

Цель: Изучить основы создания графических приложений с графикой и анимацией на языке Python, используя библиотеку tkinter.

Оборудование: ПК, ОС Windows/Linux, Python 3, среда разработки (IDLE, PyCharm, VS Code), методические указания.

Ход работы:

Часть 1. Бегущая строка (анимация текста)

В этой части мы создадим аналог проекта "Бегущая строка", используя виджеты Label (для текста) и событие after (вместо таймера TTimer). Для управления скоростью будем использовать виджет Scale (аналог TTrackBar).

Задание:

Создать приложение с бегущей строкой. Текст движется справа налево. Достигнув края, строка появляется с противоположной стороны. Скорость движения регулируется ползунком.

Ход выполнения проекта:

1. Создайте новый файл Python, например running_line.py.
2. Импортируйте необходимые модули из tkinter.

```
import tkinter as tk
```

3. Создайте главное окно приложения и установите его заголовок.

```
root = tk.Tk()
root.title("Бегущая строка")
root.geometry("400x150")  # Ширина x Высота
```

4. **Создайте виджет Label (Надпись).**
   Здесь будет отображаться движущийся текст. Важно задать фиксированную ширину, чтобы текст не менял размер метки.

```
# Переменная для хранения полного текста (буфер)
full_text = " Привет, мир! Это бегущая строка на Python. "
```

```
# Метка для отображения бегущей строки
# width задает ширину в символах, anchor="w" - выравнивание по левому краю
label = tk.Label(root, text=full_text, font=("Arial", 18), width=30, anchor="w", bg="lightgrey")
label.pack(pady=20) # pady добавляет отступы
```

5. **Создайте виджет Scale (Бегунок).**
   Он будет регулировать скорость анимации.

```
# Функция, которая будет вызываться при изменении положения ползунка
def change_speed(val):
    # Функция пока ничего не делает, кроме как получает новое значение.
    # Само значение val мы будем использовать в функции анимации.
    # Можно, например, обновлять глобальную переменную.
    global speed
    speed = int(val) # Преобразуем в целое число, так как from_ может быть float
    # print(f"Скорость изменена на {speed}") # Для отладки

# Начальная скорость (интервал в миллисекундах)
speed = 200

scale = tk.Scale(root, from_=50, to=500, orient=tk.HORIZONTAL, label="Скорость (мс):",
                 command=change_speed, length=300)
scale.set(speed) # Устанавливаем начальное положение бегунка
scale.pack()
```

6. **Создайте функцию анимации.**
   Эта функция будет имитировать событие OnTimer. Она будет изменять текст в метке, перемещая первый символ в конец, и затем планировать свой следующий запуск через определенный интервал (speed).

```
def move_text():
    # Получаем текущий текст из метки
    current_text = label.cget("text")

    # Логика движения: первый символ в конец
    if len(current_text) > 1:
        new_text = current_text[1:] + current_text[0]
    else:
        new_text = current_text # Если строка короткая, ничего не делаем

    # Обновляем текст в метке
    label.config(text=new_text)
```

```
    # Планируем следующий запуск этой же функции через `speed` миллисекунд
    # Используем глобальную переменную speed, которая обновляется ползунком
    global speed
    root.after(speed, move_text)
```

7. **Запустите анимацию.**
   Нужно вызвать функцию move_text один раз, чтобы она запустила цикл.

```
# Запускаем первый цикл анимации
root.after(speed, move_text)
```

8. **Запустите главный цикл обработки событий.**

```
root.mainloop()
```

Полный код части 1:

```
import tkinter as tk

def change_speed(val):
    global speed
    speed = int(val)

def move_text():
    current_text = label.cget("text")
    if len(current_text) > 1:
        new_text = current_text[1:] + current_text[0]
        label.config(text=new_text)
    global speed
    root.after(speed, move_text)

root = tk.Tk()
root.title("Бегущая строка")
root.geometry("400x150")

full_text = " Привет, мир! Это бегущая строка на Python. "
label = tk.Label(root, text=full_text, font=("Arial", 18), width=30, anchor="w", bg="lightgrey")
label.pack(pady=20)

speed = 200
scale = tk.Scale(root, from_=50, to=500, orient=tk.HORIZONTAL,
label="Скорость (мс):",
                 command=change_speed, length=300)
scale.set(speed)
scale.pack()
```

```
root.after(speed, move_text)
root.mainloop()
```

Часть 2. Графические методы и процедуры (простой графический редактор)

В этой части мы создадим аналог простейшего графического редактора. Вместо компонентов Lazarus (TImage, TPanel) будем использовать виджеты tkinter.Canvas (холст), Button (кнопки) и диалог выбора цвета colorchooser.

Задание:

Создать приложение, позволяющее рисовать произвольные линии на холсте при нажатой левой кнопке мыши. Реализовать выбор цвета пера и цвета фона (заливки), а также кнопку для очистки холста.

Ход выполнения проекта:

1. Создайте новый файл Python, например simple_paint.py.
2. Импортируйте модули.

   ```
   import tkinter as tk
   from tkinter import colorchooser
   ```

3. **Создайте главное окно и виджеты.**
   Разместим панель с элементами управления (Frame) и холст для рисования (Canvas). Панель с кнопками привяжем к верхней части, а холст заполнит остальное пространство.

   ```
   root = tk.Tk()
   root.title("Рисовалка на Python")
   root.geometry("600x400")

   # --- Панель инструментов (аналог Panel) ---
   toolbar = tk.Frame(root, bg='lightgrey', height=50)
   toolbar.pack(side=tk.TOP, fill=tk.X)

   # --- Холст для рисования (аналог Image) ---
   # bg - цвет фона по умолчанию
   canvas = tk.Canvas(root, bg='white')
   canvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)
   ```

4. **Настройка инструментов (переменные состояния).**
   Нам нужно хранить текущий цвет пера и последнюю позицию мыши для рисования непрерывных линий.

   ```
   python
   ```

```
# Текущий цвет пера (по умолчанию черный)
current_pen_color = 'black'

# Переменная для хранения последней позиции мыши
last_x = None
last_y = None
```

5. **Реализация рисования (обработчики событий мыши).**
   В tkinter события мыши привязываются к виджету с помощью метода bind.

- <Button-1>: Нажата левая кнопка мыши (аналог MouseDown).
- <B1-Motion>: Движение мыши с зажатой левой кнопкой (аналог MouseMove с проверкой нажатой кнопки).
- <ButtonRelease-1>: Отпускание левой кнопки (нужно, чтобы сбросить последнюю точку и не рисовать линию от места предыдущего отпускания).

```
def start_draw(event):
    """Вызывается при нажатии левой кнопки мыши."""
    global last_x, last_y
    last_x = event.x
    last_y = event.y
    # Можно также просто переместить "перо", не рисуя точку.
    # Рисование начнется при первом движении.

def draw(event):
    """Вызывается при движении мыши с зажатой левой кнопкой."""
    global last_x, last_y
    x, y = event.x, event.y
    if last_x is not None and last_y is not None:
        # Рисуем линию от предыдущей точки к текущей
        canvas.create_line(last_x, last_y, x, y,
                           fill=current_pen_color,   # Цвет линии
                           width=2,                  # Толщина линии
                           capstyle=tk.ROUND,        # Скругленные концы
                           smooth=True)              # Сглаживание
    # Обновляем последнюю точку
    last_x, last_y = x, y

def reset_draw(event):
    """Вызывается при отпускании левой кнопки мыши."""
    global last_x, last_y
    last_x, last_y = None, None

# Привязываем события к холсту
canvas.bind("<Button-1>", start_draw)
canvas.bind("<B1-Motion>", draw)
canvas.bind("<ButtonRelease-1>", reset_draw)
```

6. **Реализация выбора цвета пера.**
   Создадим кнопку на панели инструментов, которая будет открывать диалог выбора цвета. Выбранный цвет сохраним в переменную current_pen_color.

```
def choose_pen_color():
    global current_pen_color
    # Открываем диалог выбора цвета. Возвращает кортеж (RGB, HEX)
    color_code = colorchooser.askcolor(title="Выберите цвет пера",
                                       initialcolor=current_pen_color)
    # colorchooser.askcolor возвращает ((R,G,B), '#RRGGBB') или (None, None)
при отмене
    if color_code[1]: # Проверяем, что цвет выбран (HEX код не пустой)
        current_pen_color = color_code[1]
        # Изменяем цвет кнопки, чтобы было видно текущий цвет (опционально)
        pen_color_btn.config(bg=current_pen_color)

# Кнопка для выбора цвета пера
pen_color_btn = tk.Button(toolbar, text="Цвет пера",
command=choose_pen_color, bg=current_pen_color)
pen_color_btn.pack(side=tk.LEFT, padx=2, pady=5)
```

7. **Реализация заливки фона (кнопка "Фон").**
   Эта функция должна изменить цвет фона холста. В tkinter для этого используется метод config холста.

```
def choose_bg_color():
    color_code = colorchooser.askcolor(title="Выберите цвет фона",
                                       initialcolor=canvas.cget('bg'))
    if color_code[1]:
        # Меняем цвет фона холста
        canvas.config(bg=color_code[1])

# Кнопка для выбора цвета фона
bg_color_btn = tk.Button(toolbar, text="Фон", command=choose_bg_color)
bg_color_btn.pack(side=tk.LEFT, padx=2, pady=5)
```

8. **Реализация очистки холста (кнопка "Очистить").**
   Чтобы стереть всё нарисованное, можно удалить все объекты с холста методом delete("all") и, опционально, сбросить цвет фона на белый.

```
def clear_canvas():
    canvas.delete("all") # Удаляем все линии и фигуры

# Кнопка для очистки
clear_btn = tk.Button(toolbar, text="Очистить", command=clear_canvas)
clear_btn.pack(side=tk.LEFT, padx=2, pady=5)
```

9. **Запустите главный цикл.**

```
root.mainloop()
```

Полный код части 2:

```
import tkinter as tk
from tkinter import colorchooser

# --- Функции-обработчики ---
def choose_pen_color():
    global current_pen_color
    color_code = colorchooser.askcolor(title="Выберите цвет пера",
initialcolor=current_pen_color)
    if color_code[1]:
        current_pen_color = color_code[1]
        pen_color_btn.config(bg=current_pen_color)

def choose_bg_color():
    color_code = colorchooser.askcolor(title="Выберите цвет фона",
initialcolor=canvas.cget('bg'))
    if color_code[1]:
        canvas.config(bg=color_code[1])

def clear_canvas():
    canvas.delete("all")

def start_draw(event):
    global last_x, last_y
    last_x, last_y = event.x, event.y

def draw(event):
    global last_x, last_y
    x, y = event.x, event.y
    if last_x is not None and last_y is not None:
        canvas.create_line(last_x, last_y, x, y, fill=current_pen_color,
                           width=2, capstyle=tk.ROUND, smooth=True)
    last_x, last_y = x, y

def reset_draw(event):
    global last_x, last_y
    last_x, last_y = None, None

# --- Создание окна и виджетов ---
root = tk.Tk()
root.title("Рисовалка на Python")
root.geometry("600x400")

toolbar = tk.Frame(root, bg='lightgrey', height=50)
toolbar.pack(side=tk.TOP, fill=tk.X)
```

```
canvas = tk.Canvas(root, bg='white')
canvas.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# --- Переменные состояния ---
current_pen_color = 'black'
last_x, last_y = None, None

# --- Привязка событий ---
canvas.bind("<Button-1>", start_draw)
canvas.bind("<B1-Motion>", draw)
canvas.bind("<ButtonRelease-1>", reset_draw)

# --- Создание кнопок ---
pen_color_btn = tk.Button(toolbar, text="Цвет пера",
command=choose_pen_color, bg=current_pen_color)
pen_color_btn.pack(side=tk.LEFT, padx=2, pady=5)

bg_color_btn = tk.Button(toolbar, text="Фон", command=choose_bg_color)
bg_color_btn.pack(side=tk.LEFT, padx=2, pady=5)

clear_btn = tk.Button(toolbar, text="Очистить", command=clear_canvas)
clear_btn.pack(side=tk.LEFT, padx=2, pady=5)

# --- Запуск приложения ---
root.mainloop()
```

Вопросы для самоконтроля (адаптированы под Python):

1. Что такое tkinter и для чего он используется?
2. Какие основные виджеты (классы) библиотеки tkinter вы знаете?
3. Как в tkinter обрабатываются события (например, клики мыши)?
4. Для чего используется метод after у виджетов tkinter? Чем он аналогичен в Lazarus?
5. Что такое холст (Canvas) в tkinter и какие основные методы для рисования примитивов он предоставляет (create_line, create_rectangle, create_oval и т.д.)?
6. Как в tkinter открыть стандартный диалог выбора цвета?
7. Объясните назначение переменных last_x и last_y в программе-рисовалке. Почему важно сбрасывать их при отпускании кнопки мыши?
8. Чем отличается использование глобальных переменных в функциях от локальных?

Лабораторная работа 12. Разработка проектов с обработкой событий на Python

Тема: Обработка событий, анимация, работа с таймером и воспроизведение звука.

Цель: Изучить основы создания интерактивных приложений с анимацией и звуковыми эффектами на языке Python с использованием библиотеки tkinter.

Оборудование: ПК, ОС Windows/Linux, Python 3, среда разработки, методические указания.

Ход работы:

Часть 1. Программа "Светофор"

В этой части мы создадим модель работы светофора с анимацией переключения сигналов. Вместо компонентов Lazarus (TShape, TTimer) будем использовать холст Canvas для рисования кругов и метод after для анимации.

Задание:

Смоделировать работу светофора. При запуске окна светофор должен быть "пустым" (все лампочки серые или неактивные). После нажатия кнопки "Пуск" лампочки начинают переключаться с заданным интервалом. После нажатия кнопки "Стоп" все лампочки гаснут. Скорость переключения задается пользователем в поле ввода.

Ход выполнения проекта:

1. Создайте новый файл Python, например traffic_light.py.
2. Импортируйте необходимые модули.

```
import tkinter as tk
from tkinter import messagebox
```

3. Создайте главное окно и настройте его.

```
root = tk.Tk()
root.title("Светофор")
root.geometry("300x400")
root.resizable(False, False)
```

4. **Создадим холст для рисования светофора.**

Вместо трех отдельных фигур, мы нарисуем три круга на одном холсте.

```
# Холст для светофора (аналог Panel с фигурами)
canvas = tk.Canvas(root, width=150, height=350, bg='lightgrey', highlightthickness=2, highlightbackground='black')
canvas.pack(pady=10)

# Рисуем корпус светофора (прямоугольник)
canvas.create_rectangle(25, 20, 125, 330, fill='darkgrey', outline='black', width=2)

# Рисуем три круга (изначально серые/неактивные)
# Сохраним их ID, чтобы потом менять цвет
red_light = canvas.create_oval(35, 40, 115, 120, fill='grey', outline='black', width=2)
yellow_light = canvas.create_oval(35, 130, 115, 210, fill='grey', outline='black', width=2)
green_light = canvas.create_oval(35, 220, 115, 300, fill='grey', outline='black', width=2)

# Словарь для соответствия номера сигнала и его цвета
light_colors = {0: 'red', 1: 'yellow', 2: 'green'}
# Текущий активный сигнал (0 - красный, 1 - желтый, 2 - зеленый)
current_light = 0
```

5. **Создаем элементы управления.**

   Нам понадобятся: метка, поле ввода и две кнопки.

```
# Рамка для элементов управления
control_frame = tk.Frame(root)
control_frame.pack(pady=10)

# Метка и поле для ввода скорости
tk.Label(control_frame, text="Скорость (мс):").grid(row=0, column=0, padx=5)

speed_var = tk.StringVar(value="1000")  # По умолчанию 1 секунда
speed_entry = tk.Entry(control_frame, textvariable=speed_var, width=10)
speed_entry.grid(row=0, column=1, padx=5)

# Кнопки
start_btn = tk.Button(control_frame, text="Пуск", width=10)
start_btn.grid(row=1, column=0, pady=5)

stop_btn = tk.Button(control_frame, text="Стоп", width=10)
stop_btn.grid(row=1, column=1, pady=5)
```

6. **Создаем переменные состояния и функции.**

```
# Переменная для хранения ID текущего задания after (чтобы можно было отменить)
job_id = None
# Флаг работы таймера
is_running = False

def reset_lights():
    """Выключить все лампочки (сделать серыми)"""
    canvas.itemconfig(red_light, fill='grey')
    canvas.itemconfig(yellow_light, fill='grey')
    canvas.itemconfig(green_light, fill='grey')

def set_light(color_index):
    """Включить лампочку с указанным индексом (0-2)"""
    reset_lights()
    if color_index == 0:
        canvas.itemconfig(red_light, fill='red')
    elif color_index == 1:
        canvas.itemconfig(yellow_light, fill='yellow')
    elif color_index == 2:
        canvas.itemconfig(green_light, fill='green')
```

7. **Функция анимации (обработчик таймера).**

```
def next_light():
    """Переключение на следующий сигнал светофора"""
    global current_light, job_id, is_running

    if not is_running:
        return

    # Включаем текущий сигнал
    set_light(current_light)

    # Переходим к следующему сигналу (циклически 0->1->2->0)
    current_light = (current_light + 1) % 3

    # Планируем следующее переключение
    try:
        interval = int(speed_var.get())
        if interval < 100:
            interval = 100  # Минимальный интервал 100 мс
    except ValueError:
        interval = 1000
        speed_var.set("1000")
```

```
        messagebox.showwarning("Ошибка", "Введите корректное число. Используется 1000 мс")

    job_id = root.after(interval, next_light)
```

8. **Обработчики кнопок.**

```
def start_timer():
    """Обработчик кнопки Пуск"""
    global current_light, is_running, job_id

    # Если уже запущено, ничего не делаем
    if is_running:
        return

    # Останавливаем предыдущий таймер, если был
    if job_id is not None:
        root.after_cancel(job_id)
        job_id = None

    # Сбрасываем на красный свет
    current_light = 0
    is_running = True

    # Запускаем первый цикл
    try:
        interval = int(speed_var.get())
        if interval < 100:
            interval = 100
    except ValueError:
        interval = 1000
        speed_var.set("1000")

    job_id = root.after(interval, next_light)

def stop_timer():
    """Обработчик кнопки Стоп"""
    global is_running, job_id

    # Останавливаем таймер
    if job_id is not None:
        root.after_cancel(job_id)
        job_id = None

    is_running = False
```

```
    # Гасим все лампочки
    reset_lights()

# Привязываем обработчики к кнопкам
start_btn.config(command=start_timer)
stop_btn.config(command=stop_timer)
```

9. **Инициализация при запуске.**

```
# При запуске все лампочки выключены
reset_lights()
```

10. **Запуск главного цикла.**

```
root.mainloop()
```

Полный код части 1:

```
import tkinter as tk
from tkinter import messagebox

def reset_lights():
    canvas.itemconfig(red_light, fill='grey')
    canvas.itemconfig(yellow_light, fill='grey')
    canvas.itemconfig(green_light, fill='grey')

def set_light(color_index):
    reset_lights()
    if color_index == 0:
        canvas.itemconfig(red_light, fill='red')
    elif color_index == 1:
        canvas.itemconfig(yellow_light, fill='yellow')
    elif color_index == 2:
        canvas.itemconfig(green_light, fill='green')

def next_light():
    global current_light, job_id, is_running

    if not is_running:
        return

    set_light(current_light)
    current_light = (current_light + 1) % 3

    try:
```

```
            interval = int(speed_var.get())
            if interval < 100:
                interval = 100
        except ValueError:
            interval = 1000
            speed_var.set("1000")
            messagebox.showwarning("Ошибка", "Введите корректное число. Используется 1000 мс")

        job_id = root.after(interval, next_light)

    def start_timer():
        global current_light, is_running, job_id

        if is_running:
            return

        if job_id is not None:
            root.after_cancel(job_id)
            job_id = None

        current_light = 0
        is_running = True

        try:
            interval = int(speed_var.get())
            if interval < 100:
                interval = 100
        except ValueError:
            interval = 1000
            speed_var.set("1000")

        job_id = root.after(interval, next_light)

    def stop_timer():
        global is_running, job_id

        if job_id is not None:
            root.after_cancel(job_id)
            job_id = None

        is_running = False
        reset_lights()

    # Создание окна
    root = tk.Tk()
    root.title("Светофор")
```

```
root.geometry("300x400")
root.resizable(False, False)

# Холст для светофора
canvas = tk.Canvas(root, width=150, height=350, bg='lightgrey', highlightthickness=2, highlightbackground='black')
canvas.pack(pady=10)

# Рисуем корпус и лампочки
canvas.create_rectangle(25, 20, 125, 330, fill='darkgrey', outline='black', width=2)
red_light = canvas.create_oval(35, 40, 115, 120, fill='grey', outline='black', width=2)
yellow_light = canvas.create_oval(35, 130, 115, 210, fill='grey', outline='black', width=2)
green_light = canvas.create_oval(35, 220, 115, 300, fill='grey', outline='black', width=2)

# Элементы управления
control_frame = tk.Frame(root)
control_frame.pack(pady=10)

tk.Label(control_frame, text="Скорость (мс):").grid(row=0, column=0, padx=5)
speed_var = tk.StringVar(value="1000")
speed_entry = tk.Entry(control_frame, textvariable=speed_var, width=10)
speed_entry.grid(row=0, column=1, padx=5)

start_btn = tk.Button(control_frame, text="Пуск", command=start_timer, width=10)
start_btn.grid(row=1, column=0, pady=5)

stop_btn = tk.Button(control_frame, text="Стоп", command=stop_timer, width=10)
stop_btn.grid(row=1, column=1, pady=5)

# Переменные состояния
current_light = 0
job_id = None
is_running = False

# Инициализация
reset_lights()

# Запуск
root.mainloop()
```

Часть 2. Воспроизведение звука

В этой части мы создадим приложение для воспроизведения звуковых файлов формата WAV. Для этого будем использовать библиотеку playsound (самый простой способ) или pygame (более функциональный).

Установка необходимых библиотек

Перед началом работы установите библиотеку playsound:

bash

pip install playsound

Или для более продвинутого варианта с pygame:

bash

pip install pygame

Задание:

Создать приложение, которое воспроизводит звуковой файл формата WAV при нажатии на кнопку.

Ход выполнения проекта (вариант с playsound):

1. Создайте новый файл Python, например sound_player.py.
2. Импортируйте модули.

```
import tkinter as tk
from tkinter import filedialog, messagebox
from playsound import playsound
import threading
import os
```

3. Создайте главное окно.

```
root = tk.Tk()
root.title("Воспроизведение звука")
root.geometry("400x200")
```

4. **Создайте переменные для хранения пути к файлу.**

```
# Переменная для хранения пути к звуковому файлу
sound_file_path = None
is_playing = False
```

5. **Функция для выбора файла.**

```
def select_file():
    global sound_file_path
    file_path = filedialog.askopenfilename(
        title="Выберите WAV файл",
        filetypes=[("WAV files", "*.wav"), ("All files", "*.*")]
    )
    if file_path:
        sound_file_path = file_path
        file_label.config(text=f"Файл: {os.path.basename(file_path)}")
        play_btn.config(state=tk.NORMAL)
```

6. **Функция для воспроизведения звука.**

playsound блокирует выполнение программы до окончания воспроизведения. Чтобы интерфейс не зависал, запустим воспроизведение в отдельном потоке.

```
def play_sound_thread():
    global is_playing
    try:
        is_playing = True
        play_btn.config(state=tk.DISABLED, text="Играет...")
        stop_btn.config(state=tk.NORMAL)

        # Воспроизводим звук (блокирующая операция)
        playsound(sound_file_path)

    except Exception as e:
        messagebox.showerror("Ошибка", f"Не удалось воспроизвести
звук:\n{e}")
    finally:
        # После окончания воспроизведения
        is_playing = False
        play_btn.config(state=tk.NORMAL, text="Воспроизвести")
        stop_btn.config(state=tk.DISABLED)

def play_sound():
    if sound_file_path and not is_playing:
        # Запускаем в отдельном потоке, чтобы не блокировать GUI
        thread = threading.Thread(target=play_sound_thread)
        thread.daemon = True  # Поток завершится при закрытии программы
        thread.start()
```

7. **Функция остановки (для playsound сложно реализовать, поэтому сделаем заглушку).**

playsound не предоставляет простого способа остановки. Для демонстрации мы просто покажем сообщение. В следующем варианте с pygame остановка будет работать.
python

```
def stop_sound():
    messagebox.showinfo("Информация", "Библиотека playsound не поддерживает остановку.\n"
```

8. **Создаем интерфейс.**

```
# Кнопка выбора файла
select_btn = tk.Button(root, text="Выбрать WAV файл", command=select_file, width=20)
select_btn.pack(pady=10)

# Метка с именем выбранного файла
file_label = tk.Label(root, text="Файл не выбран", wraplength=350)
file_label.pack(pady=5)

# Кнопка воспроизведения
play_btn = tk.Button(root, text="Воспроизвести", command=play_sound, width=20, state=tk.DISABLED)
play_btn.pack(pady=5)

# Кнопка остановки
stop_btn = tk.Button(root, text="Стоп", command=stop_sound, width=20, state=tk.DISABLED)
stop_btn.pack(pady=5)

# Информационная метка
info_label = tk.Label(root, text="Поддерживаются только файлы WAV", fg="gray")
info_label.pack(pady=10)
```

9. **Запуск главного цикла.**

```
root.mainloop()
```

Улучшенный вариант с использованием Pygame

pygame предоставляет гораздо больше возможностей для работы со звуком, включая остановку воспроизведения.

```
import tkinter as tk
```

```
from tkinter import filedialog, messagebox
import pygame
import os

# Инициализация pygame mixer
pygame.mixer.init()

def select_file():
    global sound_file_path
    file_path = filedialog.askopenfilename(
        title="Выберите WAV или MP3 файл",
        filetypes=[("Audio files", "*.wav *.mp3"), ("All files", "*.*")]
    )
    if file_path:
        sound_file_path = file_path
        file_label.config(text=f"Файл: {os.path.basename(file_path)}")
        play_btn.config(state=tk.NORMAL)

def play_sound():
    global is_playing
    if sound_file_path:
        try:
            pygame.mixer.music.load(sound_file_path)
            pygame.mixer.music.play()
            is_playing = True
            play_btn.config(state=tk.DISABLED, text="Играет...")
            stop_btn.config(state=tk.NORMAL)

            # Проверяем окончание воспроизведения
            check_playback()

        except Exception as e:
            messagebox.showerror("Ошибка", f"Не удалось воспроизвести:\n{e}")

def check_playback():
    """Проверяет, продолжается ли воспроизведение"""
    global is_playing
    if is_playing and not pygame.mixer.music.get_busy():
        # Воспроизведение закончилось
        is_playing = False
        play_btn.config(state=tk.NORMAL, text="Воспроизвести")
        stop_btn.config(state=tk.DISABLED)
    else:
        # Проверяем снова через 100 мс
        root.after(100, check_playback)

def stop_sound():
```

```
    global is_playing
    if is_playing:
        pygame.mixer.music.stop()
        is_playing = False
        play_btn.config(state=tk.NORMAL, text="Воспроизвести")
        stop_btn.config(state=tk.DISABLED)

# Создание окна
root = tk.Tk()
root.title("Воспроизведение звука (pygame)")
root.geometry("450x250")

# Переменные
sound_file_path = None
is_playing = False

# Интерфейс
select_btn = tk.Button(root, text="Выбрать аудио файл", command=select_file, width=20)
select_btn.pack(pady=10)

file_label = tk.Label(root, text="Файл не выбран", wraplength=400)
file_label.pack(pady=5)

play_btn = tk.Button(root, text="Воспроизвести", command=play_sound, width=20, state=tk.DISABLED)
play_btn.pack(pady=5)

stop_btn = tk.Button(root, text="Стоп", command=stop_sound, width=20, state=tk.DISABLED)
stop_btn.pack(pady=5)

info_label = tk.Label(root, text="Поддерживаются WAV и MP3", fg="gray")
info_label.pack(pady=10)

root.mainloop()
```

Вопросы для самоконтроля (адаптированы под Python):

1. **Синтаксис наследования в Python:**
   - Как создать класс, наследующий от другого класса?
   - Что такое super() и для чего он используется?
2. **Обработка событий в tkinter:**
   - Как привязать функцию к событию (например, клику мыши)?
   - Что такое событийно-ориентированное программирование?

3. **Работа с таймером:**
   - Для чего используется метод after() в tkinter?
   - Чем он отличается от after_cancel()?
4. **Потоки в Python:**
   - Почему при использовании playsound нужно запускать воспроизведение в отдельном потоке?
   - Какие проблемы могут возникнуть при работе с потоками в GUI?
5. **Управление звуком:**
   - Какие библиотеки Python можно использовать для воспроизведения звука?
   - В чем разница между синхронным и асинхронным воспроизведением?
6. **Модификаторы доступа в Python:**
   - Как в Python обозначаются публичные, защищенные и приватные атрибуты?
   - В чем отличие от модификаторов в Pascal/Lazarus?
7. **Графические примитивы:**
   - Какие методы Canvas в tkinter используются для рисования кругов, прямоугольников, линий?
   - Как изменить цвет уже нарисованного объекта?
8. **Архитектура приложения:**
   - Как разделить логику приложения и пользовательский интерфейс?
   - Что такое MVC (Model-View-Controller) и как его можно применить в этих проектах?

Рекомендуемая литература

Основные источники:

1. Дорохова, Т. Ю. Основы алгоритмизации и программирования: учебное пособие для СПО / Т. Ю. Дорохова, И. Е. Ильина. - Саратов, Москва: Профобразование, Ай Пи Ар Медиа, 2022. - 139 с. - ISBN 978-5-4488-1531-7, 978-5-4497-1718-4. - Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование: [сайт]. - URL: https://profspo.ru/books/122426

2. Ковалёва, З. А. Основы программирования на языке PythonABC.NET. Основные управляющие структуры. Практикум / З. А. Ковалёва. - Санкт-Петербург: Лань, 2024. - 112 с. - ISBN 978-5-507-48265-8. - Текст: электронный // Лань: электронно-библиотечная система. - URL: https://e.lanbook.com/book/367460

3. Коренская, И. Н. Основы алгоритмизации и программирования на языке Паскаль. Лабораторный практикум: учебное пособие для спо / И. Н. Коренская. - 2-е изд., испр. - Санкт-Петербург: Лань, 2022. - 128 с. - ISBN 978-5-8114-9240-4. - Текст: электронный // Лань: электронно-библиотечная система. - URL: https://e.lanbook.com/book/189365

Дополнительные источники

1. Исаев, А. Л. Основы алгоритмизации и программирования на языке Python: практикум для СПО / А. Л. Исаев. - Саратов, Москва: Профобразование, Ай Пи Ар Медиа, 2023. - 127 с. - ISBN 978-5-4488-1639-0, 978-5-4497-2189-1. - Текст: электронный // Электронный ресурс цифровой образовательной среды СПО PROFобразование: [сайт]. - URL: https://profspo.ru/books/130049

2. Нагаева, И. А. Основы алгоритмизации и программирования: практикум: учебное пособие: [12+] / И. А. Нагаева, И. А. Кузнецов. – Москва; Берлин: Директ-Медиа, 2021. – 168 с.: схем. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=598404

3. Сидорова, Е. А. Основы программирования на языке VBA: учебное пособие / Е. А. Сидорова, С. П. Железняк. - Омск: ОмГУПС, 2021. - 118 с. - ISBN 978-5-949-41276-3. - Текст: электронный // Лань: электронно-библиотечная система. - URL: https://e.lanbook.com/book/190239

Интернет источники:

1. http://delphiexpert.ru/ - уроки, видеокурсы по программированию в среде Free Python и Delphi.

2. https://www.sites.google.com/site/ifizmat/prog/lazarus - Лабораторные работы по Lazarus.

3. http://intuit.valrkl.ru/course-708/index.html#ID.1.lecture - Программирование на Free Python и Lazarus.