

**Федеральное государственное образовательное
бюджетное учреждение высшего образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)**

**Кафедра анализа данных и машинного обучения
Факультета информационных технологий и анализа больших данных**

УТВЕРЖДАЮ

**Проректор по учебной и
методической работе**

_____ **Е.А. Каменева**

23.04.2024 г.

Макрушин С.В., Горохова Р.И., Ехлаков Р.С.

Практикум по программированию

Рабочая программа дисциплины

для студентов, обучающихся по направлению подготовки:

09.03.03 - Прикладная информатика,

ОП «Прикладные информационные системы в экономике и финансах»

*Рекомендовано Ученым советом
Факультета информационных технологий и анализа больших данных
(протокол № 43 от 16.04.2024 г.)*

*Одобрено заседанием
Кафедры анализа данных и машинного обучения
(протокол № 03 от 21.03.2024 г.)*

Москва 2024

Содержание

1. Наименование дисциплины	2
2. Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине.....	2
3. Место дисциплины в структуре образовательной программы	4
4. Объем дисциплины (модуля) в зачетных единицах и в академических часах с выделением объема аудиторной (лекции, семинары) и самостоятельной работы обучающихся	5
5. Содержание дисциплины, структурированное по темам (разделам) дисциплины с указанием их объемов (в академических часах) и видов учебных занятий.....	5
5.1. Содержание дисциплины.....	5
5.2. Учебно-тематический план.....	9
5.3. Содержание семинаров, практических занятий.....	10
6. Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине.....	11
6.1. Перечень вопросов, отводимых на самостоятельное освоение дисциплины, формы внеаудиторной самостоятельной работы	11
6.2. Перечень вопросов, заданий, тем для подготовки к текущему контролю	13
7. Фонд оценочных средств для проведения промежуточной аттестации обучающихся по дисциплине.....	63
8. Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины.....	73
9. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины.....	74
10. Методические указания для обучающихся по освоению дисциплины	76
11. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень необходимого программного обеспечения и информационных справочных систем	76
12. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине	77

1. Наименование дисциплины

«Практикум по программированию».

2. Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине

Код компетенции	Наименование компетенции	Индикаторы достижения компетенции	Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции
УК-10	Способность осуществлять поиск, критически анализировать, обобщать и систематизировать информацию, использовать системный подход для решения поставленных задач	<p>1. Четко описывает состав и структуру требуемых данных и информации, грамотно реализует процессы их сбора, обработки и интерпретации.</p> <p>2. Обосновывает сущность происходящего, выявляет закономерности, понимает природу вариабельности.</p> <p>3. Формулирует признак классификации, выделяет соответствующие ему группы однородных «объектов», идентифицирует общие свойства элементов этих групп, оценивает полноту результатов классификации, показывает прикладное назначение классификационных групп.</p>	<p>Знать: состав и структуру требуемых данных и информации.</p> <p>Уметь: грамотно реализовать процессы сбора, обработки и интерпретации данных и информации.</p> <p>Знать: закономерности реализуемых процессов и природу их вариабельности.</p> <p>Уметь: определять закономерности изучаемых процессов и данных, выявлять природу их вариабельности.</p> <p>Знать: особенности признаков классификации и оценки результатов классификации.</p> <p>Уметь: разрабатывать программный код, выполняющий классификацию по обозначенному признаку, ориентироваться в существующем коде, оценивать полноту результатов классификации.</p>

		<p>4. Грамотно, логично, аргументировано формирует собственные суждения и оценки. Отличает факты от мнений, интерпретаций, оценок и т.д. в рассуждениях других участников деятельности.</p> <p>5. Аргументированно и логично представляет свою точку зрения посредством и на основе системного описания.</p>	<p>Знать: основы логических рассуждений, аргументации, оценки своих суждений и рассуждений других участников.</p> <p>Уметь: проводить сравнительный анализ мнений, интерпретаций, оценок и т.д. в рассуждениях других участников деятельности.</p> <p>Знать: основы аргументированного и логического рассуждения.</p> <p>Уметь: проводить системное описание своей точки зрения, представлять ее аргументированно и логично</p>
ПКН-2	Способность разрабатывать алгоритмы и программы с использованием современных технологий программирования.	<p>1. Владеет объектно-ориентированным языком программирования на уровне знания синтаксиса и семантики, основ стандартной библиотеки.</p> <p>2. Использует инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки).</p>	<p>Знать: объектно-ориентированный язык программирования Python на уровне знания синтаксиса и семантики, основ стандартной библиотеки.</p> <p>Уметь: определять на уровне знания синтаксис и семантику, стандартные библиотеки языка Python, необходимые для решения прикладных задач.</p> <p>Знать: инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки).</p> <p>Уметь: разрабатывать программы решения задач с использованием инструментальных средств программирования (IDE, SDK, API, популярных фреймворков и библиотек).</p>

		<p>3.Организовывает кодовую базу, ориентируется в существующем коде, демонстрирует знание общепринятых соглашений и политик в области оформления кода.</p> <p>4. Проектирует текстовый, программный или графический интерфейс программной системы исходя из ее назначения.</p>	<p>Знать: особенности создания программного кода.</p> <p>Уметь: разрабатывать программный код, ориентироваться в существующем коде, применять знание общепринятых соглашений и политик в области оформления кода.</p> <p>Знать: основы проектирования различные виды интерфейса программной системы.</p> <p>Уметь: разрабатывать текстовый, программный или графический интерфейс программной системы исходя из ее назначения.</p>
--	--	--	--

3. Место дисциплины в структуре образовательной программы

Дисциплина «Практикум по программированию» относится к Общепрофессиональному циклу дисциплин по направлению подготовки 09.03.03 - Прикладная информатика, ОП «Прикладные информационные системы в экономике и финансах».

Дисциплина «Практикум по программированию» служит для закрепления и углубления знаний и навыков в области программирования, получаемых при освоении других дисциплин, изучаемых в рамках направления подготовки бакалавров 09.03.03 - Прикладная информатика.

4. Объем дисциплины (модуля) в зачетных единицах и в академических часах с выделением объема аудиторной (лекции, семинары) и самостоятельной работы обучающихся

Заочная форма обучения (ИОО)

Вид учебной работы по дисциплине	Всего (в з.ед. и часах)	Семестр 1 (в часах)	Семестр 2 (в часах)	Семестр 3 (в часах)	Семестр 4 (в часах)
Общая трудоемкость дисциплины	8 з.е. / 288 час.	72 час.	72 час.	72 час.	72 час.
<i>Контактная работа - Аудиторные занятия</i>	28	6	6	8	8
<i>Лекции</i>	-	-	-	-	-
<i>Семинары, практические занятия</i>	28	6	6	8	8
<i>Самостоятельная работа</i>	260	66	66	64	64
Вид промежуточной аттестации		зачет	зачет	зачет	зачет

5. Содержание дисциплины, структурированное по темам (разделам) дисциплины с указанием их объемов (в академических часах) и видов учебных занятий

5.1. Содержание дисциплины

Тема 1. Основы языка Python

Обработка числовой информации. Встроенные функции и модули для работы с числами. Реализация числовых алгоритмов с использованием инструкции ветвления и циклов.

Обработка текстовой информации. Функции и методы для работы со строками. Регулярные выражения.

Списки. Использование списков для хранения информации. Методы списков. Многомерные списки. Кортежи.

Словари. Типовые случаи использования словарей в программах. Методы для работы со словарями. Множества.

Тема 2. Функции и модули

Функции в Python: общая семантика. Создание функции и ее вызов. Расположение определений функций. Анонимные функции в Python. Необязательные параметры функций и сопоставление по ключам. Возвращение нескольких значений из функции. Распаковка и упаковка параметров функции. Аннотации и документирование функций. Глобальные и локальные переменные.

Глобальные и локальные переменные. Вложенные функции. Функции высшего порядка.

Создание и использование модулей и пакетов. Модули стандартной библиотеки.

Тема 3. Обработка исключений. Работа с файлами средствами языка Python

Понятие исключения. Инструкция `try ... except ... else ... finally`. Классы встроенных исключений. Инструкции `raise` и `assert`. Инструкция `with ... as`.

Работа с файлами в Python. Концепция файла в современных ОС и языках программирования. Операции с файлами: открытие/закрытие файла, чтение и записи и другие методы для работы с файлами. Инструкция `with ... as` и ее использование для файлов.

Использование текстовых файлов в программе. Двоичные файлы. Сохранение объектов в файл. Работа с файлами различных форматов: `csv`, `xlsx`.

Тема 4. Объектно-ориентированное программирование на Python

Python как объектно-ориентированный язык программирования. Базовые возможности ООП в Python: создание классов и объектов; наследование и полиморфизм; функция `super()`; проверка принадлежности к классу. Базовые типы в Python.

Методы классов и статические переменные и методы в Python. Управление доступом к атрибутам класса в Python. Динамические операции с атрибутами и интроспекция в Python. Использование специальных методов для расширенного функционала пользовательских классов. Кейс построения иерархии классов.

Тема 5. Функциональное программирование на Python

Элементы функционального программирования в Python: функции "граждане первого класса", функции высшего порядка, замыкания, функции без побочных эффектов, рекурсия. Неизменяемые структуры данных. Идиомы, распространенные в функциональных языках программирования: итераторы, последовательности, ленивые вычисления.

Декораторы в Python: использование и создание собственных декораторов.

Подход: map, filter, reduce. Реализация функций map, filter, reduce в Python.

Итераторы в Python, итерируемый тип данных. Модуль itertools. Функции-генераторы и выражения-генераторы в Python.

Тема 6. Алгоритмы и структуры данных

Динамические массивы. Стеки, очереди, деки. Связные списки. Реализация связных списков на языке Python. Бинарные деревья. Использование бинарных деревьев в прикладных задачах. Двоичное дерево поиска.

Асимптотическая оценка сложности алгоритма. Алгоритмы сортировки и поиска. Бинарный поиск. Простые методы сортировки: обменные сортировки, сортировка выбором, сортировка вставками. Сортировка Шелла. Быстрая сортировка. Сортировка слиянием.

Тема 7. Паттерны проектирования

Основные принципы объектно-ориентированного проектирования приложений. Принцип абстракции. Уменьшение зависимости. Основные паттерны проектирования: интерфейс, делегирование. Порождающие шаблоны: фабричный метод, абстрактная фабрика, строитель. Структурные шаблоны: адаптер, мост, декоратор, фасад. Поведенческие шаблоны: цепочка обязанностей, команда, посредник, наблюдатель, состояние, стратегия, посетитель, шаблонный метод.

Тема 8. Программирование графических интерфейсов

Событийно-ориентированное программирование. События и обработчики событий. События мыши и клавиатуры. Основные библиотеки графических интерфейсов в Python: tkinter, PyQt, PyGTK. Главный цикл программы. Основные элементы управления: кнопки, ползунки, поля ввода. Работа с графикой.

Тема 9. Системное программирование на Python

Чтение и запись файлов. Работа с путями в Windows и Linux. Обход папок. Работа с архивами. Работа с офисными форматами. Работа со структурированными данными. Основные форматы хранения данных: CSV, XML, JSON.

Тема 10. Сетевое программирование на Python

Получение и разбор HTML-страниц. Библиотеки HTML-парсинга. Сокеты. Клиент-серверные приложения. Обращение к внешним API. Многопоточность. Загруженность сетевой структуры. Библиотеки многопоточности и многопроцессности. Отправка и получение электронных писем.

Тема 11. Тестирование программ на Python

Основные виды тестирования программного обеспечения. Модульное и интеграционное тестирование. Понятие регрессии. Фиксирование и формализация требований. Библиотеки автоматизированного тестирования. Написание автоматических модульных тестов по техническому заданию.

Разработка через тестирование. Проектирование через тестирование. Методология TDD.

Тема 12. Документирование и развертывание программ на Python

Основные приемы документирования программного кода. Соглашения о стиле документирования кода. Написание программной документации в формате docstring. Языки форматирования документации: ReST, Markdown. Автоматическая сборка документации с использованием Sphinx. Экспорт документации в популярные форматы.

5.2. Учебно-тематический план

№ п/п	Наименование тем (разделов) дисциплины	Трудоемкость в часах				Формы текущего кон- троля успеваемости	
		Всего	*Контактная работа - Аудиторная работа		Самостоя- тельная работа		
			Об- щая, в т.ч.:	Лекции			Семинары, практиче- ские заня- тия
1.	Основы языка Python	25	2	-	2	21	Опрос, про- верка выпол- ненных зада- ний
2.	Функции и модули	25	2	-	2	21	Опрос, про- верка выпол- ненных зада- ний
3.	Обработка исклю- чений. Работа с файлами сред- ствами языка Python	25	2	-	2	21	Опрос, про- верка выпол- ненных зада- ний
4.	Объектно-ориен- тированное про- граммирование на Python	25	2	-	2	21	Опрос, про- верка выпол- ненных зада- ний
5.	Функциональное программирова- ние на Python	25	2	-	2	21	Опрос, про- верка выпол- ненных зада- ний
6.	Алгоритмы и структуры данных	27	2	-	2	23	Опрос, про- верка выпол- ненных зада- ний
7.	Паттерны проек- тирования	27	2	-	2	23	Опрос, про- верка выпол- ненных зада- ний
8.	Программирова- ние графических интерфейсов	27	2	-	2	23	Опрос, про- верка выпол- ненных зада- ний
9.	Системное про- граммирование на Python	27	4	-	4	23	Опрос, про- верка выпол- ненных зада- ний
10.	Сетевое програм- мирование на Python	27	2	-	2	23	Опрос, про- верка выпол- ненных зада- ний

11.	Тестирование программ на Python	27	2	-	2	23	Опрос, проверка выполненных заданий
12.	Документирование и развертывание программ на Python	27	4	-	4	23	Опрос, проверка выполненных заданий
	В целом по дисциплине	288	28	-	28	260	Согласно учебному плану: проектные работы
	Итого в %		10	-	100	90	

*объем контактной работы в очно-заочной/заочной формах обучения и индивидуальных учебных планах определяется соответствующими учебными планами. Темы, реализуемые в виде контактной работы, определяются преподавателем самостоятельно, исходя из уровня их сложности.

5.3. Содержание семинаров, практических занятий

Наименование тем (разделов) дисциплины	Перечень вопросов для обсуждения на семинарских, практических занятиях, рекомендуемые источники из разделов 8,9 (указывается раздел и порядковый номер источника)	Формы проведения занятий
1. Основы языка Python	Решение задач с использованием инструкции ветвления. Решение задач с использованием циклов. Создание и обработка списков. Многомерные списки. Обработка текстовой информации. Использование словарей и множеств. Совместное использование различных типов данных. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
2. Функции и модули	Создание и использование функций. Анонимные функции. Функции высшего порядка. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
3. Обработка исключений. Работа с файлами средствами языка Python	Работа с текстовыми файлами. Работа с двоичными файлами. Сохранение объектов в файл. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
4. Объектно-ориентированное программирование на Python	Создание классов и объектов. Наследование и полиморфизм. Специальные методы классов. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере

5.Функциональное программирование на Python	Функции map, filter, reduce, any, all. Декораторы. Итераторы. Функции генераторы. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
6.Алгоритмы и структуры данных	Создание связанных списков. Использование стеков и очередей при решении задач. Алгоритмы сортировки и поиска. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
7.Паттерны проектирования	Создание примеров программных реализаций для распространенных шаблонов проектирования. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
8.Программирование графических интерфейсов	Создание графического приложения с использованием встроенных возможностей языка и библиотек <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
9.Системное программирование на Python	Автоматизация рутинных административных задач. Создание Web-парсера. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
10. Сетевое программирование на Python	Создание многопоточного многопользовательского многофункционального сервера на сокетах. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
11.Тестирование программ на Python	Написание модульных тестов по описанию. Разработка программы по тестам. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере
12. Документирование и развертывание программ на Python	Документирование программы по описанию и исходному коду. Экспорт документации в формате сайта. <i>Основная литература - [8.1 - 8.3]</i> <i>Дополнительная литература - [8.1-8.2]</i>	Интерактивная форма, работа на компьютере

6. Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине

6.1. Перечень вопросов, отводимых на самостоятельное освоение дисциплины, формы внеаудиторной самостоятельной работы

Наименование тем (разделов) дисциплины	Перечень вопросов, отводимых на самостоятельное освоение	Формы внеаудиторной самостоятельной работы
1. Основы языка Python	Функции модуля math, random, copy. Регулярные выражения. Кортежи. Множества.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
2. Функции и модули	Создание и использование модулей и пакетов.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
3. Обработка исключений. Работа с файлами средствами языка Python	Работа с файлами различных форматов: csv, docx, xlsx.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
4. Объектно-ориентированное программирование на Python	Специальные методы классов.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
5. Функциональное программирование на Python	Функции any, all, zip.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
6. Алгоритмы и структуры данных	Бинарные деревья. Использование бинарных деревьев в прикладных задачах. Двоичное дерево поиска.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
7. Паттерны проектирования	Внедрение зависимости, инверсия управления.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
8. Программирование графических интерфейсов	Создание игр. Библиотека PyGame.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
9. Системное программирование на Python	Создание административных скриптов.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.

10.Сетевое программирование на Python	Создание веб-сервиса.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
11.Тестирование программ на Python	Экстремальное программирование.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.
12.Документирование и развертывание программ на Python	Непрерывная интеграция.	Работа с учебной литературой и документацией. Решение задач с использованием Jupyter Notebook. Выполнение домашних заданий.

6.2. Перечень вопросов, заданий, тем для подготовки к текущему контролю

Примерные задания проектных работ

Задание 1

1. Реализовать программу, с которой можно играть в логическую игру «Быки и коровы» (описание правил игры: <http://робомозг.рф/Articles/BullsAndCowsRules>). Программа загадывает число, пользователь вводит очередной вариант отгадываемого числа, программа возвращает количество быков и коров и в случае выигрыша игрока сообщает о победе и завершается. Сама программа НЕ ходит, т.е. не пытается отгадать число загаданное игроком.

Взаимодействие с программой производится через консоль, при запросе данных от пользователя программа сообщает, что ожидает от пользователя и проверяет корректность ввода.

2. Реализовать программу, при помощи которой 2 игрока могут играть в «Крестики-нолики» на поле 3 на 3. Взаимодействие с программой производится через консоль. Игровое поле изображается в виде трех текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных

от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты новой отметки на поле) и проверяет корректность ввода. Программа должна уметь автоматически определять, что партия окончена, и сообщать о победе одного из игроков или о ничьей. Сама программа НЕ ходит, т.е. не пытается ставить крестики и нолики с целью заполнить линию.

3. Реализовать программу, при помощи которой 2 игрока могут играть в игру «Супер ним». Правила игры следующие. На шахматной доске в некоторых клетках случайно разбросаны фишки или пуговицы. Игроки ходят по очереди. За один ход можно снять все фишки с какой-либо горизонтали или вертикали, на которой они есть. Выигрывает тот, кто заберет последние фишки. (описание правил игры: <https://www.iqfun.ru/articles/super-nim.shtml>)

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты новой отметки на поле) и проверяет корректность ввода. Программа должна уметь автоматически определять, что партия окончена, и сообщать о победе одного из игроков. Сама программа НЕ ходит, т.е. не пытается выбирать строки или столбцы с целью победить в игре.

4. Реализовать программу, с которой можно играть в игру «19». Правила игры следующие. Нужно выписать подряд числа от 1 до 19: в строчку до 9, а потом начать следующую строку, в каждой клетке по 1 цифре (не числу (см пример по ссылке)). Затем игроку необходимо вычеркнуть парные цифры или дающие в сумме 10. Условие - пары должны находиться рядом или через зачеркнутые цифры по горизонтали или по вертикали. После того как все возможные пары вычеркнуты, оставшиеся цифры переписываются в конец таблицы. Цель - полностью вычеркнуть все цифры. (описание правил игры: <http://podelki-fox.ru/igry-dlya-detey-na-bumage-s-chislami/>)

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде трех текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять, что нужно выписать новые строки с цифрами и то, что партия окончена. Сама программа НЕ ходит, т.е. не пытается выбирать пары цифр с целью окончить игру.

5. Реализовать программу, при помощи которой 3 игрока могут играть в игру «Лоскутное одеяло». Правила игры следующие. На поле, имеющем размер 4 на 5 клеток за один ход каждый игрок должен заполнить одну клетку своим символом. Игрок старается, чтобы его символы были как можно дальше друг от друга. В ходе игры ведется подсчет очков: за каждое соседство клеток с одинаковыми символами игроку, владельцу символа добавляется одно штрафное очко. Соседними считаются клетки, имеющие общую сторону или расположенные наискосок друг от друга. Выигрывает тот, у кого в конце игры меньше всего штрафных очков.

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 4 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять количество штрафных очков и окончание партии и ее победителя.

Сама программа НЕ ходит, т.е. не пытается заполнять клетки символами с целью выиграть игру.

6. Реализовать программу, при помощи которой 2 игрока могут играть в игру «Клондайк». Правила игры следующие. Игра ведётся на игровом поле размером 10 на 10 клеток. Игроки по очереди выставляют в любую свободную клетку по отметке, и тот игрок, после чьего хода получилась цепочка длиной

хотя бы в 3 отметке, проигрывает. При этом в цепочке считаются как свои отметки, так и отметки соперника, у игровых фишек как бы нет хозяина. Цепочка - это ряд фишек, следующая фишка в котором примыкает к предыдущей с любого из 8-ми направлений. (описание правил игры: <https://www.iqfun.ru/printable-puzzles/klondike-igra.shtml>)

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 10 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять окончание партии и ее победителя.

Сама программа НЕ ходит, т.е. не пытается ставить в клетки отметки с целью выиграть игру.

7. Реализовать программу, при помощи которой 2 игрока могут играть в игру «Максит». Правила игры следующие. В клетках квадрата 3 на 3 пишутся случайные числа из диапазона от 1 до 9. Начинаящий выбирает любое понравившееся ему число и вычеркивает его, прибавляя к своей сумме. Второй игрок может выбрать любое из оставшихся чисел того столбца, в котором первый игрок делал свой предыдущий ход. Он тоже вычеркивает выбранное число, прибавляя его к своей сумме. Первый игрок далее поступает аналогично, выбирая число-кандидата из той строки, в которой второй игрок ходил перед этим. Может так случиться, что у какого-то игрока не будет хода. Тогда его соперник продолжает игру, делая ход в той же строке (для первого игрока) или в том же столбце (для второго игрока), что и до этого. Игра заканчивается, когда оба играющих не имеют ходов. Результат определяется по набранным суммам, у кого она больше, тот и выиграл. При равенстве сумм фиксируется ничья. (описание правил игры: <https://www.iqfun.ru/articles/maxit.shtml>).

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 3 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять сумму очков каждого из игроков и окончание партии и ее победителя.

Сама программа НЕ ходит, т.е. не пытается вычеркивать числа с целью выиграть игру.

8. (*) Реализовать программу, при помощи которой 2 игрока могут играть в игру «Мостики». Правила игры следующие. В ходе игры каждый из игроков старается построить мост с одного своего берега на другой по камням, образующим массив 4 на 5 (4 камня вдоль берега игрока и 5 камней между берегами). У первого игрока - крестики в качестве камней и берега крестиков (левый и правый край поля), у второго игрока – нолики и берега ноликов (верхний и нижний край поля). Игру можно начинать в любой точке поля. За один ход игрок может соединить два своих соседних камня вертикальным или горизонтальным мостиком (обозначаются в текстовом режиме символами «-» и «|»). Мосты первого и второго игрока пересекаться не должны. Выигрывает тот, кто построит непрерывный мост с одного своего берега на другой. (описание правил игры: <https://www.7ya.ru/article/Chem-zanyat-rebenka-13-igr-na-liste-bumagi-so-slovami-kartinkami/>)

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 9 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода) и проверяет корректность ввода. Программа должна уметь автоматически определять недопустимые ходы (приводящие к пересечению мостов соперников) и окончание партии и ее победителя.

Сама программа НЕ ходит, т.е. не пытается строить мосты с целью выиграть игру.

9. (*) Реализовать программу, с которой можно играть в игру «Морской бой». Программа автоматически случайно расставляет на поле размером 10 на 10 клеток: 4 1-палубных корабля, 3 2-палубных корабля, 2 3-палубных корабля и 1 4-х палубный. Между любыми двумя кораблями по горизонтали и вертикали должна быть как минимум 1 незанятая клетка. Программа позволяет игроку ходить, производя выстрелы. Сама программа НЕ ходит т.е. не пытается топить корабли расставленные игроком.

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 10 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (в частности, координаты очередного «выстрела») и проверяет корректность ввода. Программа должна уметь автоматически определять потопление корабля и окончание партии и сообщать об этих событиях.

- 10.(*) Реализовать программу, с которой можно играть в игру «Пятнашки». Правила игры следующие.

Головоломка представляет собой 15 квадратных костяшек с числами от 1 до 15. Все костяшки заключены в квадратную коробку (поле) размером 4 на 4. При размещении костяшек в коробке остается одно пустое место, которое можно использовать для перемещения костяшек внутри коробки. Цель игры - упорядочить размещение чисел в коробке, разместив их по возрастанию слева направо и сверху вниз, начиная с костяшки с номером 1 в левом верхнем углу и заканчивая пустым местом в правом нижнем углу коробки.

Взаимодействие с программой производится через консоль. Игровое поле изображается в виде 4 текстовых строк и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, координаты очередного хода)

и проверяет корректность ввода. Программа должна считать количество сделанных ходов, уметь автоматически определять недопустимые ходы, окончание партии и ее победителя.

Сама программа НЕ ходит, т.е. не пытается упорядочить костяшки с целью выиграть игру.

Задание 2

Базовая часть:

Написать калькулятор для строковых выражений вида '<число> <операция> <число>', где <число> - не отрицательное целое число меньшее 100, записанное словами, например "тридцать четыре", <арифметическая операция> - одна из операций "плюс", "минус", "умножить". Результат выполнения операции вернуть в виде текстового представления числа. Пример `calc("двадцать пять плюс тринадцать")` -> "тридцать восемь"

Оформить калькулятор в виде функции, которая принимает на вход строку и возвращает строку.

1. Реализовать поддержку операции деления и остатка от деления и работу с дробными числами (десятичными дробями). Пример: `calc("сорок один и тридцать одна сотая разделить на семнадцать")` -> "два и сорок три сотых". Обращать дробную часть до тысячных включительно, если при делении получаются числа с меньшей дробной частью выполнять округление до тысячных.

Сложность: 2

2. Расширение задания 1. Реализовать поддержку десятичной дробной части до миллионных долей включительно. Реализовать корректный вывод информации о периодической десятичной дроби (период дроби вплоть до 4х десятичных знаков). Пример: `calc("девятнадцать и восемьдесят две сотых разделить на девяносто девять")` -> "ноль и двадцать сотых и ноль два в периоде".

Сложность 3

3. Реализовать текстовый калькулятор для выражения из произвольного количества операций с учетом приоритета операций. Пример: `calc("пять плюс два умножить на три минус один")` -> "десять". (Для реализации рекомендуется использовать алгоритмы основанные на польской инверсной записи см. например,
https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D1%82%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%BB%D1%8C%D1%81%D0%BA%D0%B0%D1%8F_%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C) *Сложность 3*
4. Расширение задания 3. Добавить поддержку приоритета операций с помощью скобок. Пример: `calc("скобка открывается пять плюс два скобка закрывается умножить на три минус один")` -> "двадцать". *Сложность 3*
5. Добавить возможность использования отрицательных чисел. Пример: `calc("пять минус минус один")` -> "шесть". *Сложность 1*
6. Добавить возможность оперировать с дробями (правильными и смешанными). Реализовать поддержку сложения, вычитания и умножения, дробей. Результат операций не должен представлять неправильную дробь, такие результаты нужно превращать в смешанные дроби. Пример: `calc("один и четыре пятых плюс шесть седьмых ")` -> "два и двадцать три тридцать пятых".
Сложность 3
7. Расширение задания 6. Добавить автоматическое сокращение дроби в ответе. Пример: `calc("одна шестая умножить на две третьих")` -> "одна девятая".
Сложность 1
8. Расширение задания 1. Добавить операции возведения в степень и тригонометрические операции синус, косинус, тангенс и константу пи. Допускается как минимум одна из этих функций в выражении с обычными операциями. Пример: `calc("два в степени четыре")` -> "шестнадцать". Пример: `calc("синус от пи разделить на четыре")` -> "ноли и семьсот семь тысячных".
Сложность 1 или 2

9. Добавить комбинаторные операции перестановки, размещения и сочетания.

Пример: `calc("размещений из трех по два")` -> "шесть". Сложность 1 или 2

10. Диагностировать ошибки: неправильную запись числа; неправильную последовательность чисел и операций; (задание 1) деление на ноль; (задание 3) неправильную последовательность чисел и операций; (задание 4) некорректный баланс и вложенность скобок; (задание 6) некорректную запись числа.

Сложность 1 или 2

Задание 3

Базовая часть (выполняется всеми самостоятельно!):

На базе модулей: `csv`, `pickle` и прямой работы с файлами реализовать следующий базовый функционал:

1. функций **`load_table`**, **`save_table`** по загрузке/сохранению табличных данных во внутреннее представление модуля/из внутреннего представления модуля:
 - файла формата `csv` (отдельный модуль с **`load_table`**, **`save_table`** в рамках общего пакета)
 - файла формата `pickle` (отдельный модуль с **`load_table`**, **`save_table`** в рамках общего пакета), модуль использует структуру данных для представления таблицы, удобную автору работы.
 - текстового файла (только функция **`save_table`** сохраняющая в текстовом файле представление таблицы, аналогичное выводу на печать с помощью функции **`print_table()`**).

Примечание: внутреннее представление может базироваться на словаре, где по разным ключам хранятся ключевые «атрибуты» таблицы, а значения таблицы хранятся в виде вложенных списков. Студент может выбрать другое внутреннее представление таблицы (согласовав его с преподавателем), в том числе,

студенты знакомые с ООП на Python, могут реализовать собственный класс для таблицы.

При определении API модулей максимально полно использовать возможности сигнатур функций на Python (значения по умолчанию, упаковка/распаковка, в т.ч. именованных параметров, возвращение множественных значений), интенсивно выполнять проверки и возбуждать исключительные ситуации.

2. модуля с базовыми операциями над таблицами:

- **get_rows_by_number(start, [stop], copy_table=False)** – получение таблицы из одной строки или из строк из интервала по номеру строки. Функция либо копирует исходные данные, либо создает новое представление таблицы, работающее с исходным набором данных (**copy_table=False**), таким образом изменения, внесенные через это представления будут наблюдаться и в исходной таблице.
- **get_rows_by_index(val1, ... , copy_table=False)** – получение новой таблицы из одной строки или из строк со значениями в первом столбце, совпадающими с переданными аргументами **val1, ... , valN**. Функция либо копирует исходные данные, либо создает новое представление таблицы, работающее с исходным набором данных (**copy_table=False**), таким образом изменения, внесенные через это представления будут наблюдаться и в исходной таблице.
- **get_column_types(by_number=True)** – получение словаря вида *столбец:тип_значений*. Тип значения: int, float, bool, str (по умолчанию для всех столбцов). Параметр **by_number** определяет вид значения столбец – целочисленный индекс столбца или его строковое представление.
- **set_column_types(types_dict, by_number=True)** – задание словаря вида *столбец:тип_значений*. Тип значения: int, float, bool, str (по умолчанию для всех столбцов). Параметр **by_number** определяет вид значения столбец – целочисленный индекс столбца или его строковое представление.

- **get_values(column=0)** – получение списка значений (типизированных согласно типу столбца) таблицы из столбца либо по номеру столбца (целое число, значение по умолчанию 0, либо по имени столбца)
 - **get_value(column=0)** – аналог **get_values(column=0)** для представления таблицы с одной строкой, возвращает не список, а одно значение (типизированное согласно типу столбца).
 - **set_values(values, column=0)** – задание списка значений **values** для столбца таблицы (типизированных согласно типу столбца) либо по номеру столбца (целое число, значение по умолчанию 0, либо по имени столбца).
 - **set_value(column=0)** – аналог **set_values(value, column=0)** для представления таблицы с одной строкой, устанавливает не список значений, а одно значение (типизированное согласно типу столбца).
 - **print_table()** – вывод таблицы на печать.
3. Для каждой функции должно быть реализована генерация не менее одного вида исключительных ситуаций.

Индивидуальные задания:

1. В `load_table` реализовать `load_table(file1, ...)` – поддержку загрузки таблицы, разбитой на несколько файлов (произвольное количество файлов) (для форматов `csv` и `pickle`). В случае несоответствия структуры столбцов файлов вызывать исключительную ситуацию. *Сложность 1*
2. *Расширение задания 1.* В `save_table` реализовать поддержку сохранения таблицы в разбитой на несколько файлов (произвольное количество файлов) по параметру `max_rows`, определяющему максимальное количество строк в файле. Файлы `csv` и `pickle`, полученные с помощью `save_table` должны быть совместимы с `load_table` из задания 1. *Сложность 1*
3. Реализовать функцию `concat(table1, table2)` и `split(row_number)` склеивающую две таблицы или разбивающую одну таблицу на 2 по номеру строки.

Сложность 1

4. Реализовать автоматическое определение типа столбцов по хранящимся в таблице значениям. Оформить как отдельную функцию и встроить этот функционал как опцию работы функции `load_table`. *Сложность 1 или 2*
5. Реализовать поддержку дополнительного типа значений «дата и время» на основе модуля `datetime`. *Сложность 1 или 2*
6. Добавить набор функций `add`, `sub`, `mul`, `div`, которые обеспечат выполнение арифметических операций для столбцов типа `int`, `float`, `bool`. Продумать сигнатуру функций и изменения в другие функции, которые позволят удобно выполнять арифметические операции со столбцами и присваивать результаты вычислений. Реализовать реагирование на некорректные значения с помощью генерации исключительных ситуаций. *Сложность 2*
7. По аналогии с п. 6 реализовать функции `eq (==)`, `gr (>)`, `ls (<)`, `ge (>=)`, `le (<=)`, `ne (==)`, которые возвращают список булевских значений длиной в количество строк сравниваемых столбцов. Реализовать функцию `filter_rows (bool_list, copy_table=False)` – получение новой таблицы из строк для которых в `bool_list` (длиной в количество строк в таблице) находится значение `True`. *Сложность 3*
8. Реализовать функцию `merge_tables(table1, table2, by_number=True)`: в результате слияния создается таблица с набором столбцов, соответствующих объединенному набору столбцов исходных таблиц. Соответствие строк ищется либо по их номеру (`by_number=True`) либо по значению индекса (1й столбец). При выполнении слияния возможно множество конфликтных ситуаций. Автор должен их описать и определить допустимый способ реакции на них (в т.ч. через дополнительные параметры функции и инициацию исключительных ситуаций). *Сложность 2*

9. Реализовать полноценную поддержку значения None в незаполненных ячейках таблицы. Должно работать при загрузке ячеек с пропусками значений, при операциях приводящих к появлению пустых ячеек, при работе с get и set операциями. *Сложность 1 или 2*

Задание 4

Базовая часть (выполняется всеми самостоятельно!):

Реализовать программу, которая позволяет играть в шахматы на компьютере. Взаимодействие с программой производится через консоль (базовый вариант). Игровое поле изображается в виде 8 текстовых строк, плюс строки с буквенным обозначением столбцов (см. пример на Рис. 1) и перерисовывается при каждом изменении состояния поля. При запросе данных от пользователя программа сообщает, что ожидает от пользователя (например, позицию фигуры для следующего хода белыми; целевую позицию выбранной фигуры) и проверяет корректность ввода (допускаются только ходы соответствующие правилам шахмат; поддержка рокировки, сложных правил для пешек и проверки мата вынесена в отдельные пункты). Программа должна считать количество сделанных ходов.

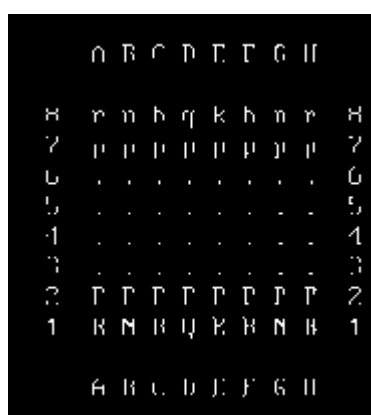


Рис. 1 Пример изображения шахматного поля в текстовом режиме

Сама программа НЕ ходит: т.е. не пытается выполнить ходы за одну из сторон, а предоставляет поочередно вводить ходы за белых и черных.

Индивидуальные задания:

Справка о шахматной нотации:

- Общая информация о шахматной нотации записи партий:
https://ru.wikipedia.org/wiki/Шахматная_нотация
- Партии в полной нотации: бесплатная база (для открытия партий нужно зарегистрироваться на ресурсе) записей партий в шахматной нотации (полной): <http://www.chessebook.com/openings.php?lan=ru&pa=pa> (для получения файлов с записью партий копируйте текст понравившихся партий в текстовый файл, скопированный текст не подвергать дополнительному редактированию и сохранить в файл).
- Партии в сокращенной нотации берем из обсуждений на kasparovchess.crestbook.com, например, из этой ветки: <http://kasparovchess.crestbook.com/threads/8210/> (для получения файлов с записью партий копируйте текст понравившихся партий, расположенных справа от блока с доской, в текстовый файл, скопированный текст не подвергать дополнительному редактированию (он во многих нюансах будет отличаться от партий с chessebook.com, так и должно быть) и сохранять файл).

1. Реализовать чтение записи шахматной партии из выбранного пользователем файла в полной нотации. После чтения должна быть возможность двигаться вперед и назад по записи партии (с соответствующим изменением на поле). Должна быть возможность в выбранной позиции перейти из режима просмотра партии в обычный режим игры.

Протестировать не менее чем на 20 реальных партиях с сайта.

Сложность 2

2. Реализовать чтение записи шахматной партии из выбранного пользователем файла в сокращенной нотации. После чтения должна быть возможность двигаться вперед и назад по записи партии (с соответствующим изменением на

поле). Должна быть возможность в выбранной позиции перейти из режима просмотра партии в обычный режим игры.

Протестировать не менее чем на 20 реальных партиях с сайта. *Сложность 3 (если пункты 1 и 2 совместно, то суммарная сложность 4)*

3. Реализовать возможность записи разыгрываемой шахматной партии в текстовый файл в полной (сокращенной, если студент выполнял задание 2) нотации. Записать партию можно на любом ходу, с историей всей партии с самого начала. Записанная партия должна корректно воспроизводиться в режиме чтения записи партии. *Сложность 2*
4. Реализовать возможность «отката» ходов. С помощью специальной команды можно возвращаться на ход (или заданное количество ходов) назад вплоть до начала партии. *Сложность 1*
5. Реализовать функцию подсказки выбора новой позиции фигуры: после выбора фигуры для хода функция визуально на поле показывает поля доступные для хода или фигуры соперника, доступные для взятия, выбранной фигурой. *Сложность 1*
6. Реализовать функцию подсказки угрожаемых фигур: она возвращает информацию о том, какие фигуры ходящего игрока сейчас находятся под боем (т.е. могут быть взяты соперником на следующий ход) и визуально выделяет их на поле. Функция отдельно указывает на наличие шаха королю. *Сложность 1*
7. Автоматически определять мат (правила определения: [https://ru.wikipedia.org/wiki/Мат_\(шахматы\)](https://ru.wikipedia.org/wiki/Мат_(шахматы))). *Сложность 2*
8. Реализовать поддержку выполнения рокировки по всем шахматным правилам (в базовой версии поддержка рокировки не обязательна). Правила рокировки см.: <https://ru.wikipedia.org/wiki/Рокировка> . *Сложность 1*
9. Реализовать поддержку для пешки сложных правил: «взятие на проходе» и замены на других фигуру при достижении крайней горизонтали (в базовой версии их поддержка не обязательна, но возможность первого хода на одну

или две горизонтали - обязательно). Подробнее о правилах см.: https://ru.wikipedia.org/wiki/Правила_шахмат . *Сложность 1*

Задание 5

Задача коммивояжера. Описание

Задача математического программирования по определению оптимального маршрута движения коммивояжера, цель которого состоит в том, чтобы посетить все объекты, записанные в задании, за кратчайший срок и с наименьшими затратами. В теории графов Задача коммивояжера - это поиск пути, связывающего два или более узла, с использованием критерия оптимальности.

Задача коммивояжера (коммивояжёр — бродячий торговец) заключается в отыскании самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешёвый, совокупный критерий и т. п.) и соответствующие матрицы расстояний, стоимости и т. п. Как правило, указывается, что маршрут должен проходить через каждый город только один раз, в таком случае выбор осуществляется среди гамильтоновых циклов.

Задача коммивояжера – это классическая комбинаторная задача теоретической информатики. Существует несколько способов решения задачи коммивояжера. Некоторые популярные решения:

Метод перебора – вычисляются все возможные пути, а затем сравниваются. Количество путей в графе, состоящем из n городов, равно $n!$. Решение при таком подходе требует больших вычислительных затрат.

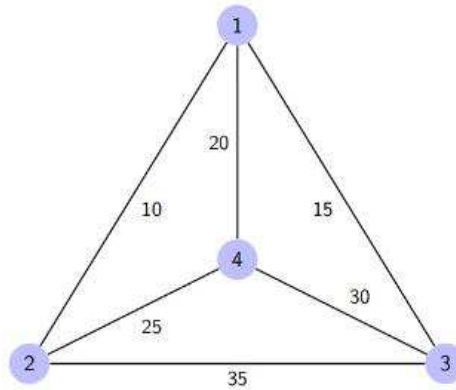
Метод ветвей и границ – проблема разбивается на подзадачи. Решение отдельных подзадач обеспечит оптимальное решение.

Динамическое программирование – метод поиска оптимальных решений путем анализа всех возможных маршрутов.

Метод ближайшего соседа – эвристически жадный подход, при котором

мы выбирается соседний узел. Вычислительно менее затратен, чем динамический подход, но не дает гарантии оптимального решения.

Задача состоит в том, чтобы найти кратчайший путь в графе с условием посещения всех узлов только один раз и возвращения в исходный город. Необходимо вывести на экран кратчайший путь.



Для приведенного выше графа оптимальным маршрутом является путь с минимальными затратами: $1 - 2 - 4 - 3 - 1$. Кратчайший маршрут будет стоить: $10 + 25 + 30 + 15 = 80$.

Пример решения методом динамического программирования:

Предположим, S — это подмножество городов, принадлежащее $\{1, 2, 3, \dots, n\}$, где $1, 2, 3, \dots, n$ — города, а i, j — два города в этом подмножестве. Теперь $Cost(i, S, j)$ определяется таким образом, как длина кратчайшего пути, посещающего узел в S , который ровно один раз имеет начальную и конечную точки как i и j соответственно. Например, стоимость $(1, \{2, 3, 4\}, 1)$ обозначает длину кратчайшего пути, где: стартовый город — 1; города 2, 3 и 4 посещаются только один раз, конечная точка — 1.

Алгоритм:

1. Установить стоимость $(i, \dots, i) = 0$, что означает, что точка старта и финиша алгоритма в i , а стоимость равна 0.
2. Когда $|S| > 1$, то $Cost(i, S, 1) = \infty$, где $i \neq 1$. Потому что изначально неизвестна стоимость проезда из города i в город 1 через другие города.

1. Жадные алгоритмы: Метод ближайшего соседа
2. Жадные алгоритмы: Метод включения ближайшего города
3. Жадные алгоритмы: Метод самого дешёвого включения
4. Метод минимального остовного дерева.
5. Метод имитации отжига
6. Метод ветвей и границ
7. Метод эластичной сети
8. Муравьиный алгоритм
9. Генетический алгоритм
10. Алгоритм динамического программирования

Задание 6

Реализация собственного пакета модулей для манипулирования плоскими фигурами

Базовые требования к функциональности программы:

Реализовать `api`, которое позволяет: генерировать, преобразовывать и визуализировать последовательность плоских полигонов, представленных в виде кортежа кортежей (например: $((0,0), (0,1), (1,1), (1,0))$) – представление для квадрата). Последовательности представлений полигонов представляют из себя итераторы (далее: последовательности полигонов). Решать задачи с использованием функционального стиля программирования, в том числе активно использовать функции из модуля `itertools` и `functools`.

1. Реализовать функцию визуализации последовательности полигонов, представленной в виде итератора (например, можно использовать визуализацию с помощью библиотеки `matplotlib`, см. пример):

https://matplotlib.org/stable/gallery/shapes_and_collections/patch_collection.html#sphx-glr-gallery-shapes-and-collections-patch-collection-py

(обязательная часть)

2. Реализовать функции, генерирующие бесконечную последовательность не пересекающихся полигонов с различающимися координатами (например, «ленту»):

1. прямоугольников (**gen_rectangle()**);
2. треугольников (**gen_triangle()**);
3. правильных шестиугольников (**gen_hexagon()**).
4. с помощью данных функций используя функции из модуля `itertools` сгенерировать 7 фигур, включающих как прямоугольники, так и треугольники и шестиугольники, визуализировать результат.

(обязательная часть)

3. Реализовать операции:

1. параллельный перенос (**tr_translate**);
2. поворот (**tr_rotate**);
3. симметрия (**tr_symmetry**);
4. гомотетия (**tr_homothety**);

которые можно применить к последовательности полигонов с помощью функции `map`.

(обязательная часть)

4. С помощью данных функций создать и визуализировать:

1. 3 параллельных «ленты» из последовательностей полигонов, расположенных под острым углом к оси x ;
2. две пересекающихся «ленты» из последовательностей полигонов, пересекающихся не в начале координат;
3. две параллельных ленты треугольников, ориентированных симметрично друг к другу;
4. последовательность четырехугольников в разном масштабе, ограниченных двумя прямыми, пересекающимися в начале координат (см. рис.).

(обязательная часть)

5. Реализовать операции:

1. фильтрации фигур, являющихся выпуклыми многоугольниками (**flt_convex_polygon**);
2. фильтрации фигур, имеющих хотя бы один угол, совпадающий с заданной точкой (**flt_angle_point**);
3. фильтрации фигур, имеющих площадь менее заданной (**flt_square**);
4. фильтрации фигур, имеющих кратчайшую сторону менее заданного значения (**flt_short_side**);
5. фильтрации выпуклых многоугольников, включающих заданную точку (внутри многоугольника) (**flt_point_inside**);
6. фильтрации выпуклых многоугольников, включающих любой из углов заданного многоугольника (**flt_polygon_angles_inside**);

которые можно применить к последовательности полигонов с помощью функции filter.

(обязательная часть: 2 пункта, 4 пункта – сложность 1; 6 пунктов – сложность 2)

6. С помощью данных функций реализовать и визуализировать:

1. фильтрацию фигур, созданных в рамках пункта 4.4; подобрать параметры так, чтобы на выходе было получено 6 фигур;
2. используя функции генерации из п. 2 и операции из п. 3 создать не менее 15 фигур, которые имеют различный масштаб и выбрать из них (подбором параметра фильтрации) не более 4х фигур, имеющих кратчайшую сторону менее заданного значения;
3. используя функции генерации из п. 2 и операции из п. 3 создать не менее 15 фигур имеющих множество пересечений и обеспечить фильтрацию пересекающихся фигур.

(обязательная часть: 1 пункт, 3 пункта – сложность 1)

7. Реализовать декораторы и продемонстрировать корректность их работы:

1. Фильтрующие многоугольники в итераторах среди аргументов функции, работающие на основе функций из 5:

**@flt_convex_polygon, @flt_angle_point, @flt_square,
@flt_short_side, @flt_point_inside, @flt_polygon_angles_inside;**

2. Преобразующие многоугольники в итераторах среди аргументов функции, работающие на основе функций из 3: **@tr_translate, @tr_rotate, @tr_symmetry, @tr_homothety ;**

(обязательная часть: 1 пункта, 5 пунктов – сложность 1)

8. Реализовать функции и продемонстрировать их корректность:

1. поиск угла, самого близкого к началу координат (**agr_origin_nearest**);
2. поиск самой длинной стороны многоугольника (**agr_max_side**);
3. поиск самой маленькой площади многоугольника (**agr_min_area**);
4. расчет суммарного периметра (**agr_perimeter**);
5. расчет суммарной площади (**agr_area**).

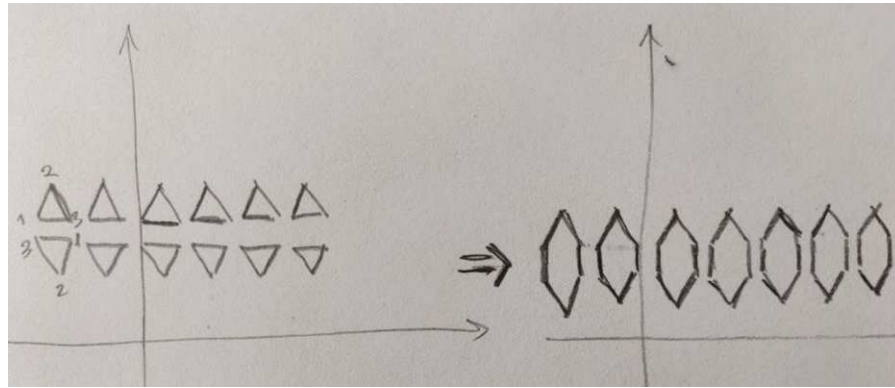
которые можно применить к последовательности полигонов с помощью функции `functools.reduce` .

(3 пункта – сложность 1; 5 пунктов – сложность 2)

9. Реализовать функции и продемонстрировать пример их работы (если возможно, с визуализацией):

1. склейки полигонов в одну последовательность полигонов из нескольких последовательностей полигонов **zip_polygons(iterator1, iterator2, [iterator3, ...])**. Пример: **zip_polygons([((1,1), (2,2), (3,1)), ((11,11), (12,12), (13,11))], [((1,-1), (2,-2), (3,-1)), ((11,-11), (12,-12), (13,-11))])** -> **[((1,1), (2,2), (3,1), (1,-1), (2,-2), (3,-1)), ((11,11), (12,12), (13,11), (11,-11), (12,-12), (13,-11))]** .

Альтернативный пример (визуализация):



2. генерации **count_2D()** параметры: (start1, start2), [(step1, step2)], результаты: (start1, start2), (start1+step1, start2+step2), (start1+2*step1, start2+2*step2)
3. склейки полигонов в одну последовательность полигонов из нескольких последовательностей **zip_tuple(iterator1, iterator2)** . Пример: **zip_tuple([(1,1), (2,2), (3,3), (4,4)], [(2,2), (3,3), (4,4), (5,5)], [(3,3), (4,4), (5,5), (6,6)])** -> ((1,1), (2,2), (3,3)), ((2,2), (3,3), (4,4)), ((3,3), (4,4), (5,5)), ((5,5), (6,6), (7,7))

(3 пункта – сложность 1)

Для каждого студента формируется комплексное задание из сочетания пунктов. Суммарная сложность комплексного задания должна быть не менее 5, как минимум одна задача из комплекта должна стоить дороже 1. Приветствуется выполнение заданий с суммарной сложностью более 5 (рекомендуется отмечать такие решения дополнительными баллами).

Сложность комплексного задания может быть скорректирована преподавателем в зависимости от сочетания пунктов между собой.

Задание 7-8

Простая анимация

Цель работы: Основная цель решения задач из данного списка - освоить приемы работы с графической библиотекой, а именно: построение графических примитивов, создание анимации, управление главным циклом работы программы.

Порядок выполнения:

Задания в этой лабораторной работе предусматривают индивидуальное самостоятельное выполнение. Преподаватель распределяет задания между студентами случайным образом. На семинаре преподаватель может по запросу студентов пояснить задание в части требований к выполнению или стоящей за заданием математической концепцией.

Все задания подразумевают творческий подход к их выполнению. Задание описывает только общую канву программы. В процессе выполнения студент может столкнуться с рядом дизайнерских решений - выбор скорости анимации, размера элементов, цветового решения и так далее. Все это ложится на студента как на разработчика программы. Преподаватель вправе оценить удачные решения и снизить оценку за неудачные.

По умолчанию предусматривается использование библиотеки `tkinter`. По желанию студента и по согласованию с преподавателем можно выбрать другой подобный инструмент решения задачи (`pygame`, `p5` или аналогичные графические библиотеки).

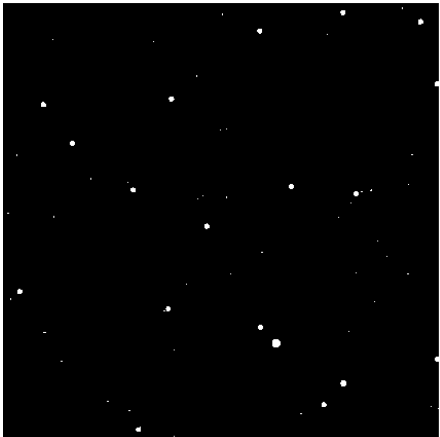
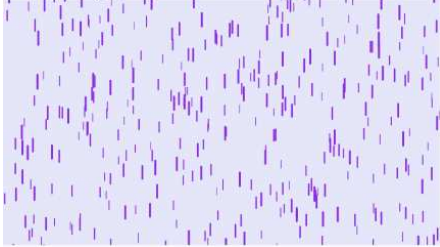
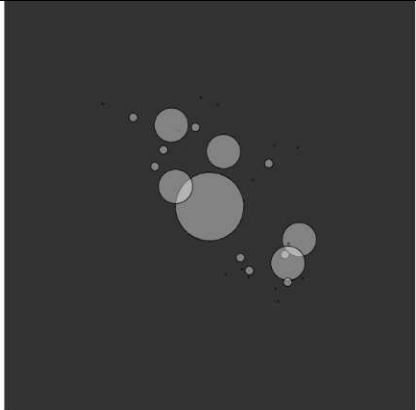
К каждой задаче предлагаются дополнительные задания (пронумерованы), которые расширяют или углубляют данное задание. Студент может самостоятельно предложить расширение функционала программы. Дополнительные задания оцениваются преподавателем с учетом их сложности.

Дополнительные задания предусматривают самостоятельное освоение студентом некоторых инструментов - обработки событий, работы с элементами пользовательского интерфейса, работу с мышью и клавиатурой. Следует помнить, что все эти навыки пригодятся для выполнения дальнейших заданий.

После самостоятельного выполнения задания преподаватель может попросить студента провести публичную демонстрацию выполненного задания,

включая устные комментарии написанного хода и алгоритмических решений, примененных студентом при решении задачи.

Задания для выполнения

<p>Звездное поле</p> <p>В произвольном месте на экране появляются звезды, которые движутся по направлению от центра экрана. Должно складываться ощущение движения наблюдателя сквозь звездное поле. Сделайте звезды разного размера и цветов.</p> <ol style="list-style-type: none"> 1. Реализуйте движение в 3D, где размер звезды зависит от расстояния до нее. 2. Скорость движения регулируется с клавиатуры. 3. Мышкой можно регулировать направление движения. 	
<p>Дождь</p> <p>На экране постоянно генерируются контрастные “капли”, равномерно движущиеся вниз.</p> <ol style="list-style-type: none"> 1. Поэкспериментируйте с плотностью генерации капель. 2. Капли должны создавать ощущение трехмерности - дальние движутся медленнее и меньше в размерах. 	
<p>Солнечная система</p> <p>В центре экрана находится неподвижное солнце. Вокруг него вращаются несколько планет по круговым орбитам с разной скоростью и на разном расстоянии.</p> <ol style="list-style-type: none"> 1. Попробуйте добавить прозрачность. 2. У планет могут быть собственные спутники. 3. Добавьте пояс астероидов. 	

Задание 9

Сложная анимация

Цель работы

Цель выполнения задач из данного списка - применить полученные навыки работы с графикой и анимацией для выполнения более сложных проектов. Каждая задача предполагает продумывание структуры программы, то есть этап проектирования.

Порядок выполнения

Задания в этой проектной работы предусматривают индивидуальное самостоятельное выполнение. Преподаватель распределяет задания между студентами случайным образом. На семинаре преподаватель может по запросу студентов пояснить задание в части требований к выполнению или стоящей за заданием математической концепцией.

Все задания подразумевают творческий подход к их выполнению. Задание описывает только общую канву программы. В процессе выполнения студент может столкнуться с рядом дизайнерских решений - выбор скорости анимации, размера элементов, цветового решения и так далее. Все это ложится на студента как на разработчика программы. Преподаватель вправе оценить удачные решения и снизить оценку за неудачные.

Многие задания из данного списка предполагают освоение какой-то математической концепции, необходимой для понимания задания или путей его выполнения. Этап исследования - важный и необходимый при написании прикладного программного обеспечения.

Строго рекомендуется при выполнении заданий из данного сборника использовать объектно-ориентированный подход. Вы должны следить за оформлением кода - соблюдением стилевых соглашений, необходимости комментировать код, его читаемости. Дополнительным плюсом будет использование системы контроля версий.

К каждой задаче предлагаются дополнительные задания (пронумерованы), которые расширяют или углубляют данное задание. Студент может самостоятельно предложить расширение функционала программы. Дополнительные задания оцениваются преподавателем с учетом их сложности.

Дополнительные задания предусматривают самостоятельное освоение студентом некоторых инструментов - обработки событий, работы с элементами пользовательского интерфейса, работу с мышью и клавиатурой. Следует помнить, что все эти навыки пригодятся для выполнения дальнейших заданий.

После самостоятельного выполнения задания преподаватель может попросить студента провести публичную демонстрацию выполненного задания, включая устные комментарии написанного хода и алгоритмических решений, примененных студентом при решении задачи.

1. *Фейерверки (152)*. На экране случайно генерируются фейерверки. Сам фейерверк и его частицы имеют ограниченное время жизни. Добавьте генерацию фейерверков разных цветов и прозрачность. Поэкспериментируйте с гравитацией частиц.

2. *Притяжение (113)*. Реализуйте систему частиц, каждая из которых притягивается к определенному месту на экране. Одновременно, каждая частица отталкивается от текущих координат мышки. Должно получиться, что курсор "разбрасывает" частицы от себя. Нарисуйте местами притяжения частиц какой-нибудь рисунок или надпись. Реализуйте импорт рисунка из внешнего файла.

3. *Анимация круга (78)* С помощью полярных координат реализующее превращение круга в треугольник как на приведенной анимации

4. *Папоротник Барнсли (48)* При помощи интерактивного алгоритма постройте из точек фрактальную фигуру. Манипулируя параметрами уравнения, получите другие виды фрактальных растений.

Задание 10

Простые игры

Цель работы

Цель выполнения работ из данного списка - применить на практике навыки работы с двумерной анимацией в разработке простых аркадных игр и получить навыки командной работы, проектирования и управления длительной разработкой.

Порядок выполнения

Задания в этой лабораторной работе предусматривают самостоятельное выполнение в малых группах (2 человека). Преподаватель распределяет задания между студентами случайным образом. На семинаре преподаватель может по запросу студентов пояснить задание в части требований к выполнению или стоящей за заданием математической концепцией.

Все задания подразумевают творческий подход к их выполнению. Задание описывает только общую канву программы. В процессе выполнения студент может столкнуться с рядом дизайнерских решений - выбор скорости анимации, размера элементов, цветового решения и так далее. Все это ложится на студента как на разработчика программы. Преподаватель вправе оценить удачные решения и снизить оценку за неудачные.

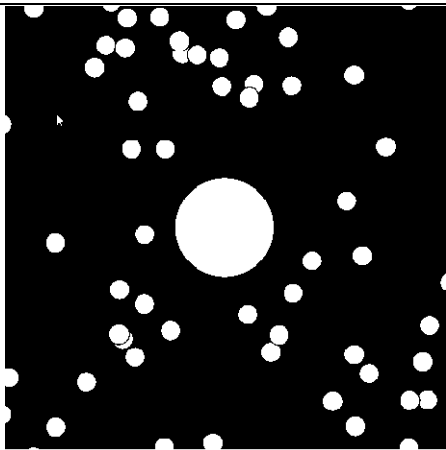

Задания из данного списка построены на классических играх. К каждому заданию прилагается анимация, иллюстрирующая механику игры. При разработке игр рекомендуется предварительно ознакомиться с существующими реализациями, правилами игры и их вариациями. Следует в первую очередь реализовать именно классический вариант данной игры. При желании в механику и правила игры можно внести изменения.

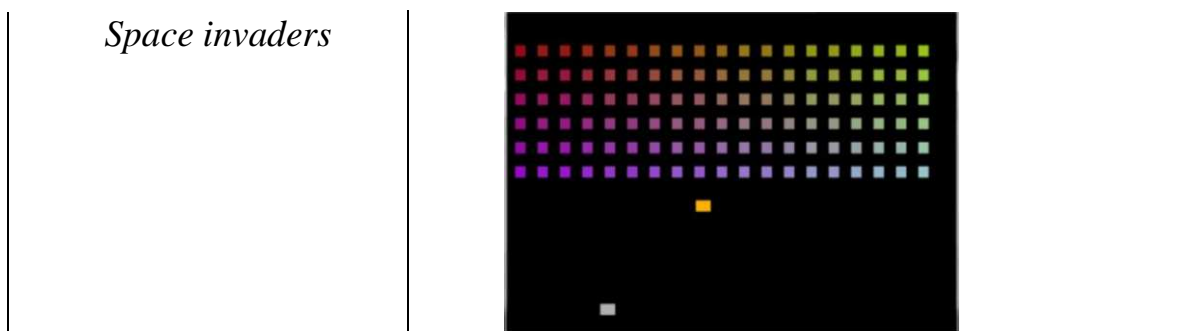
Разработка программ из данного списка может предполагать работу с графическими элементами - фоновыми изображениями, спрайтами, моделями. По возможности следует создать эти элементы самостоятельно (это может стать от-

дельной ролью студента в проекте), либо найти свободно распространяемые варианты.

Строго рекомендуется при выполнении заданий из данного сборника использовать объектно-ориентированный подход. Вы должны следить за оформлением кода - соблюдением стилевых соглашений, необходимости комментировать код, его читаемости. Строго рекомендуется использование системы контроля версий для организации коллективной работы.

После самостоятельного выполнения задания преподаватель может попросить студента провести публичную демонстрацию выполненного задания, включая устные комментарии написанного хода и алгоритмических решений, примененных студентом при решении задачи.

<i>Agar.io</i>	
<i>Блек Джек</i>	



Задание 11

Приёмы работы в современных системах контроля версий.

Цель работы: познакомиться на практике с основными приемами работы в современных системах контроля версий.

Задачи для выполнения:

1. Работа с системой контроля версий Git.
2. Работа GUI git клиент GitKraken.
3. Работа с удаленными репозиториями и GitHub.

Требования к отчету

Отчет должен содержать:

- Описание каждого упражнения и его цели.
- Пошаговое объяснение реализации задачи, включая описание используемых функций и методов.
- Анализ полученных результатов и выводы.
- Любые трудности, с которыми студенты столкнулись при выполнении задания, и способы их решения.
- Размышления о том, как полученные знания могут быть применены в практических проектах.

Упражнение 1. Работа с системой контроля версий Git.

Для выполнения работы необходимо воспользоваться любым консольным клиентом системы контроля версий Git, подходящую для рабочей операционной системы. Для Windows подойдет, например, дистрибутив [git-scm](#). Этот же пакет

включает в себя и консольную утилиту git, необходимую для выполнения следующих работ. Для Linux существует, например, пакет [Git Cola](#).

Для написания сообщений коммита необходимо придерживаться следующего общепринятого правила: в первой строке сообщения следует кратко описать произведенные изменения; если необходимо подробное описание, состоящее из многих строк, то его приводят, отступив от первой строки одну пустую. Помните, что заголовок описания коммита - это то, что будете видеть вы и ваши коллеги в истории изменений проекта.

Для работы в терминале изучите шпаргалки по основным командам git.

Главные команды, которые вам понадобятся это:

git init - создает репозиторий системы контроля версий в данной директории;

git add - добавляет указанный файл под версионный контроль;

git add . - добавляет все файлы текущей директории под версионный контроль;

git status - показывает состояние рабочей директории по сравнению с последним сохраненным состоянием:

git commit -m "(message)" - сохраняет текущее состояние рабочей директории как новое состояние (создает новый коммит); новый коммит получает сообщение, переданное как параметр;

git commit -am "(message)" - создает новый коммит и автоматически включает в него все изменившиеся отслеживаемые файлы;

git log - выводит историю коммитов репозитория;

git branch - показывает список веток репозитория;

git branch <branchname> - создает новую ветку на основе текущего состояния с переданным названием;

git checkout -b <branchname> - создает новую ветку и автоматически делает ее текущей;

`git merge <branchname>` - сливает изменения, сделанные в ветке с переданным названием в текущую;

`git branch -d <branchname>` - удаляет ветку с переданным названием;

`git clone (repo URL)` - клонирует удаленный репозиторий, находящийся по переданному адресу в текущую директорию; доступ обычно осуществляется по протоколам HTTP либо SSH;

`git fetch` - считывает изменения в удаленном репозитории, отсутствующие в локальной копии;

`git pull` - считывает новые изменения в удаленном репозитории и сливает их в соответствующие локальные ветки;

`git push --all` - отправляет изменения, сделанные в локальном репозитории в удаленный;

1.1. Установить на компьютер графический клиент Git.

Подсказка по командам

```
• pwd — покажи, в какой я папке;
• ls — покажи файлы в папке, где я сейчас;
• cd first-project — перейди в папку first-project ;
• cd first-project/html — перейди в папку html, находящуюся в папке first-project ;
• cd .. — перейди на уровень выше в родительскую папку;
• cd ~ — перейди в домашнюю директорию (у нас это /Users/stas_basov );
• mkdir second-project — в текущей папке создай папку с именем second-project ;
• rm about.html — удали файл about.html ;
• rmdir images — удали папку images ;
• rm -r second-project — удали папку second-project и всё, что она содержит;
• touch index.html — создай файл index.html в текущей папке;
• touch index.html style.css script.js — если нужно создать несколько файлов, их имена можно вводить через пробел.
```

1.2. Создайте в своей домашней папке (или в любой другой на ваш выбор) каталог, который будет содержать файлы нового программного проекта.

1.3. Выберите тематику программы, которую собираетесь написать.

- 1.4. Инициализируйте в этой директории репозиторий гит при помощи команды `git init`. Обратите внимание на появление в этой папке скрытой подпапки с названием `.git`. Если вы ее не видите, то скорее всего, у вас отключено отображение скрытых папок.

```
C:\Users\zxfursed>cd C:\Users\zxfursed\PycharmProjects  
  
C:\Users\zxfursed\PycharmProjects>git init 123  
Initialized empty Git repository in C:/Users/zxfursed/PycharmProjects/123/.git/
```

- 1.5. Выполните в репозитории команду `git status`. Проинтерпретируйте полученное сообщение.

```
C:\Users\zxfursed\PycharmProjects\123>git status  
On branch master  
  
No commits yet  
  
nothing to commit (create/copy files and use "git add" to track)
```

Пока что нет ни одного коммита

- 1.6. Создайте файл для исходного текста программы. Выполните команду `git status`.

```
C:\Users\zxfursed\PycharmProjects\123>git status  
On branch master  
  
No commits yet  
  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
    main.py  
  
nothing added to commit but untracked files present (use "git add" to track)
```

Теперь виден файл.

«Гит» называет эти файлы "Untracked files" — неотслеживаемые файлы. Это значит, что «Гит» их видит, но, если попытаться сохранить их версию сейчас, «Гит» этого не сделает. Для этого файлы нужно подготовить к сохранению. За это отвечает команда `git add`. При её использовании нужно указать имя файла, судьбу которого мы хотим зафиксировать в текущем виде.

- 1.7. Добавьте созданный файл под версионный контроль при помощи команды `git add`. Еще раз выполните `git status`.

```
C:\Users\zxfursed\PycharmProjects\123>git add main.py

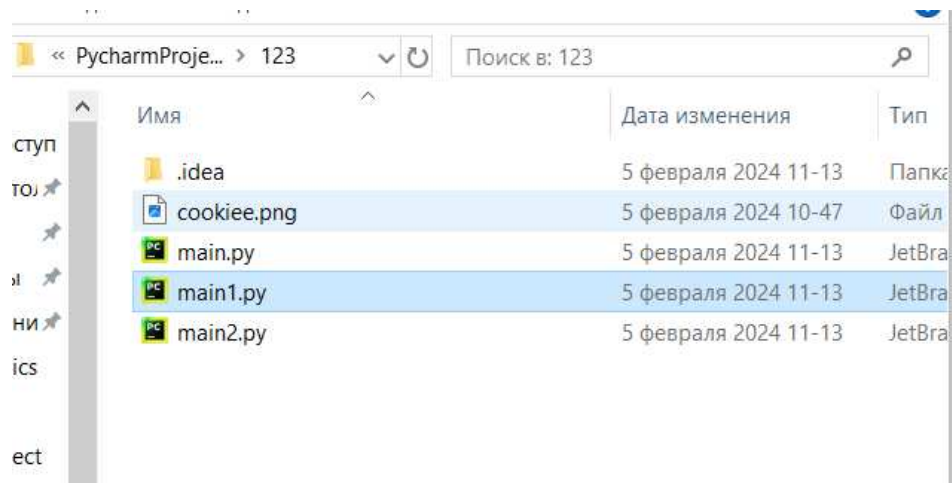
C:\Users\zxfursed\PycharmProjects\123>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   main.py

C:\Users\zxfursed\PycharmProjects\123>
```

- 1.8. Когда все файлы готовы к сохранению, сделаем наш первый коммит — зафиксируем все сделанные изменения в «боевой версии». Делается это командой `git commit` с опцией `-m`. После `-m` идёт название коммита в кавычках.
- 1.9. Сделайте еще несколько коммитов. Выполните команду `git log` для просмотра истории коммитов.
- 1.10. Сделайте так, чтобы при коммите измененные файлы автоматически добавлялись в коммит.
- 1.11. Добавьте еще несколько файлов с исходным текстом программы.



- 1.12. Добавьте все новые файлы под версионный контроль одной командой.

```

C:\Users\zxfursed\PycharmProjects\123>git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        main1.py
        main2.py

nothing added to commit but untracked files present (use "git add" to track)
C:\Users\zxfursed\PycharmProjects\123>git add -A

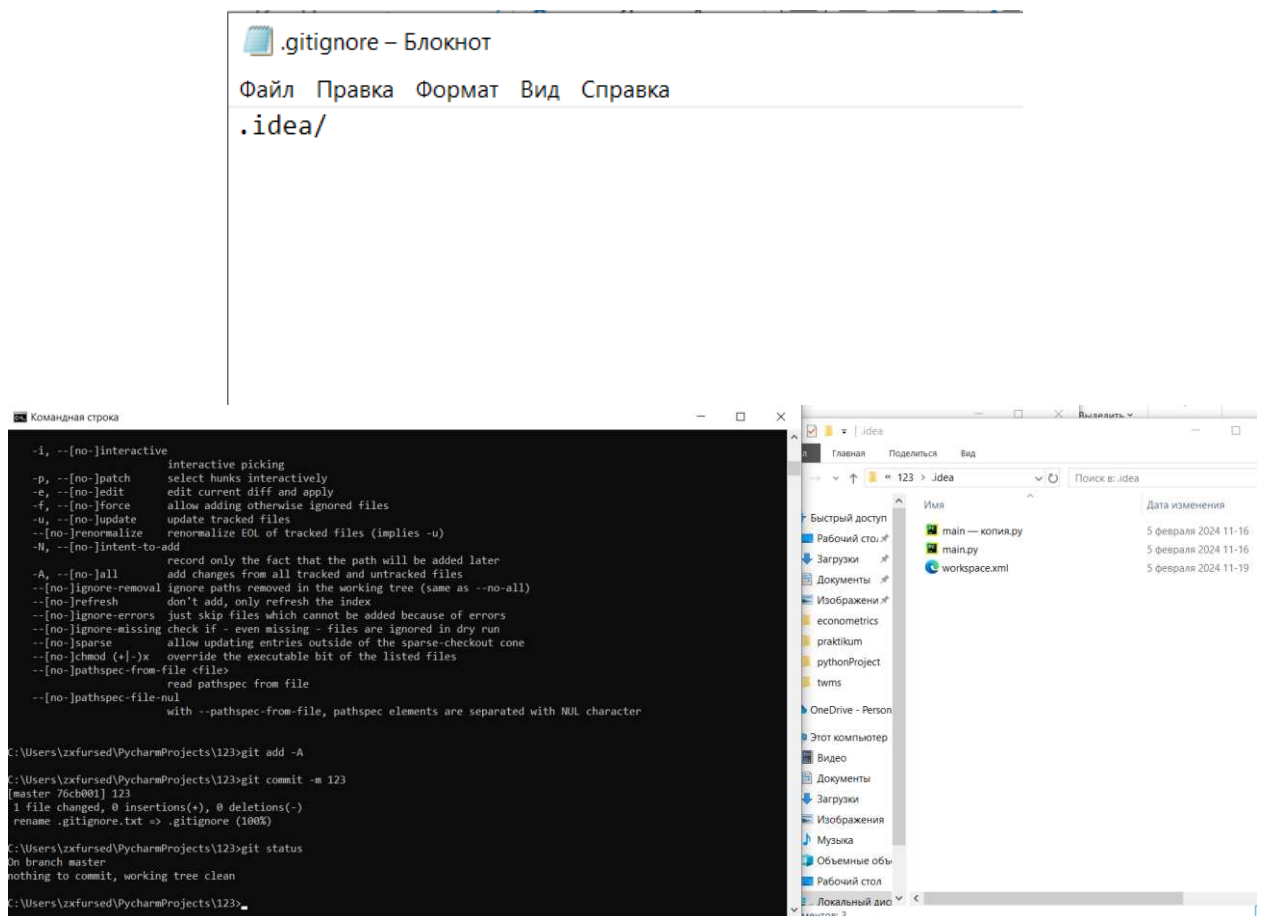
C:\Users\zxfursed\PycharmProjects\123>git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   main1.py
        new file:   main2.py

C:\Users\zxfursed\PycharmProjects\123>git commit -m newfiles
[master 0d823f4] newfiles
 2 files changed, 56 insertions(+)
 create mode 100644 main1.py
 create mode 100644 main2.py
C:\Users\zxfursed\PycharmProjects\123>

```

1.13. Инициализируйте в рабочей директории виртуальное окружение

1.14. Добавьте созданную служебную папку в файл .gitignore. Проверьте, что они не добавляются в репозитории при добавлении новых файлов с ИСХОДНЫМ КОДОМ.



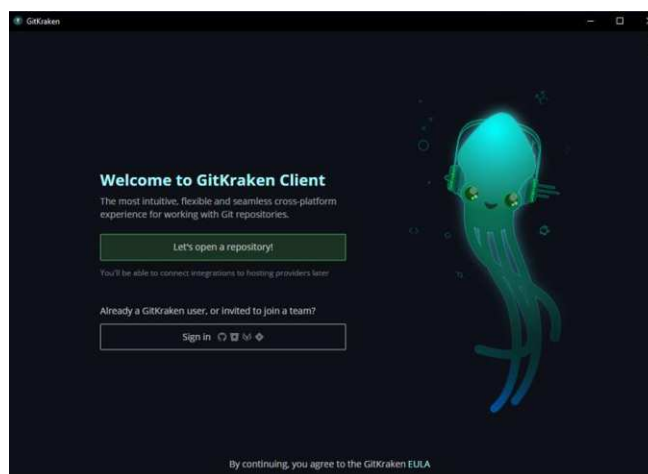
- 1.15. Создайте новую тематическую ветку git branch. Перейдите в нее с помощью git checkout. Выведите на экран список всех веток.

```
C:\Users\zxfursed\PycharmProjects\123>git branch vetka  
C:\Users\zxfursed\PycharmProjects\123>git checkout  
C:\Users\zxfursed\PycharmProjects\123>git checkout vetka  
Switched to branch 'vetka'  
C:\Users\zxfursed\PycharmProjects\123>git branch  
master  
* vetka  
C:\Users\zxfursed\PycharmProjects\123>
```

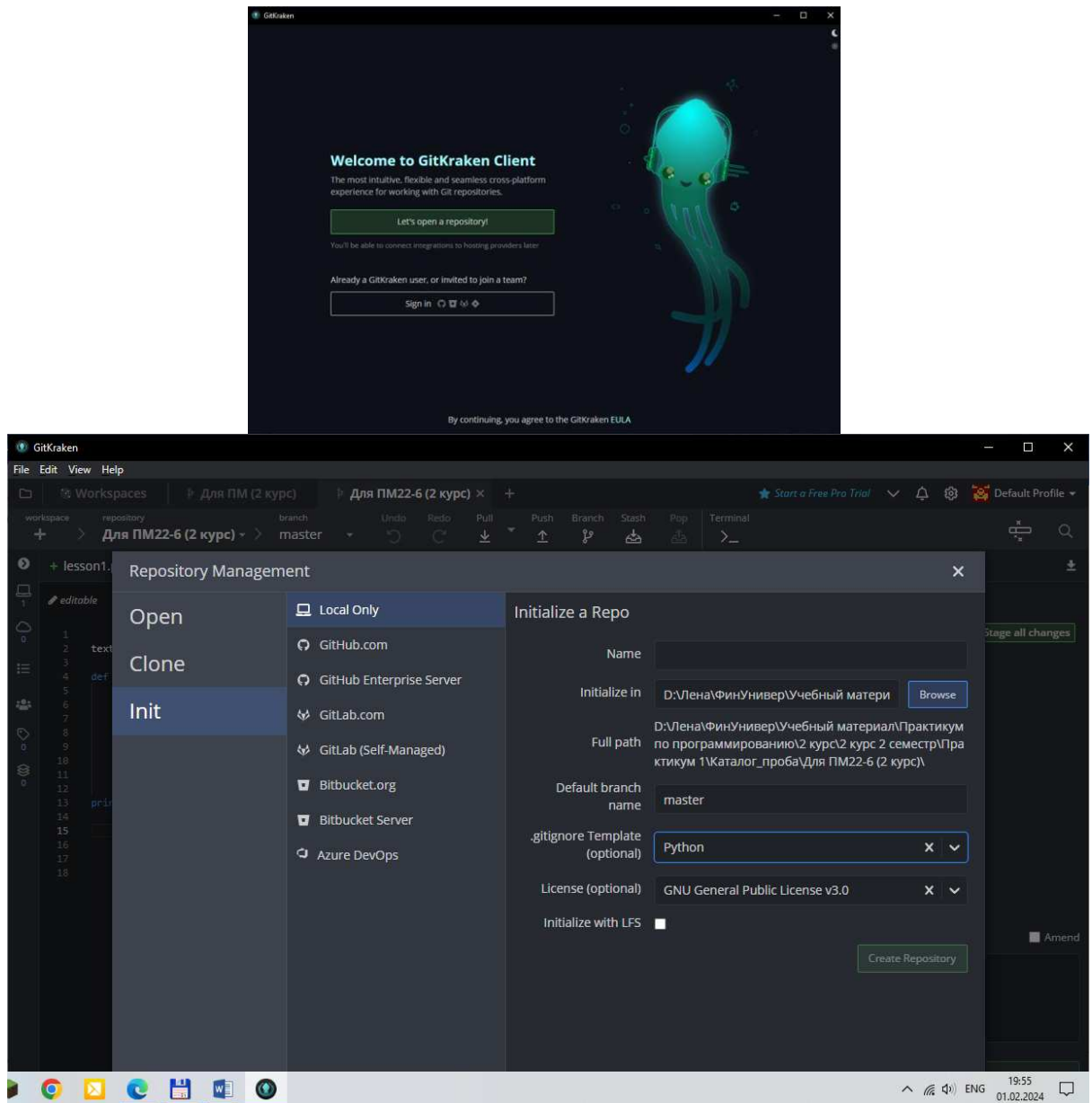
- 1.16. Сделайте несколько коммитов в основную и тематическую ветки
- 1.17. Слейте изменения в основную ветку с помощью git merge. Если произошел конфликт слияния, разрешите его и завершите слияние с помощью git commit.
- 1.18. При получении в процессе разработки программы в стабильно работающем состоянии, слейте это состояние в основную ветку и добавьте к коммиту слияния пометку с номером релиза.

Упражнение 2. Работа GUI git клиент GitKraken.

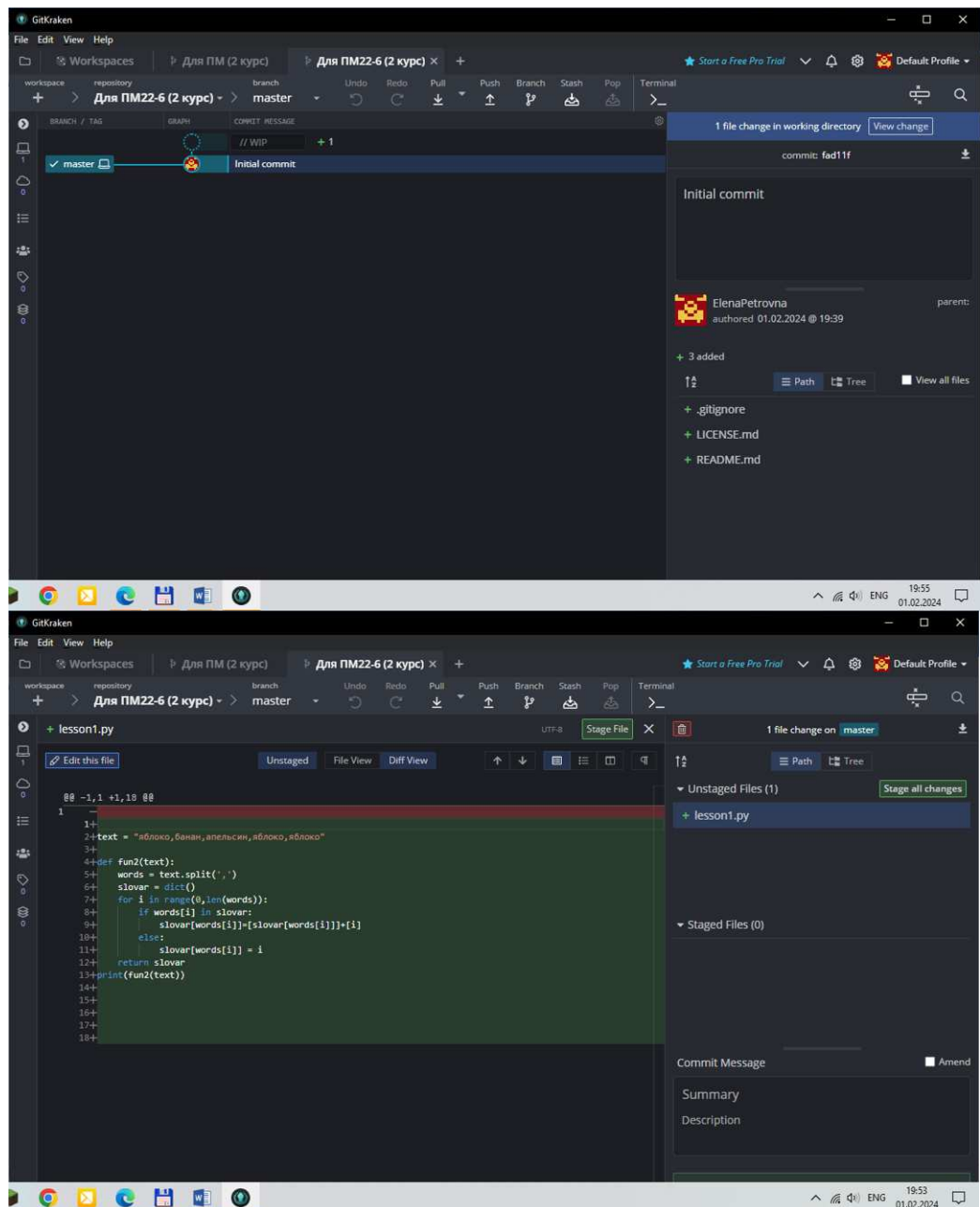
GitKraken — это кроссплатформенный, удобный и высокоэффективный GUI git клиент на Linux, Windows и macOS.



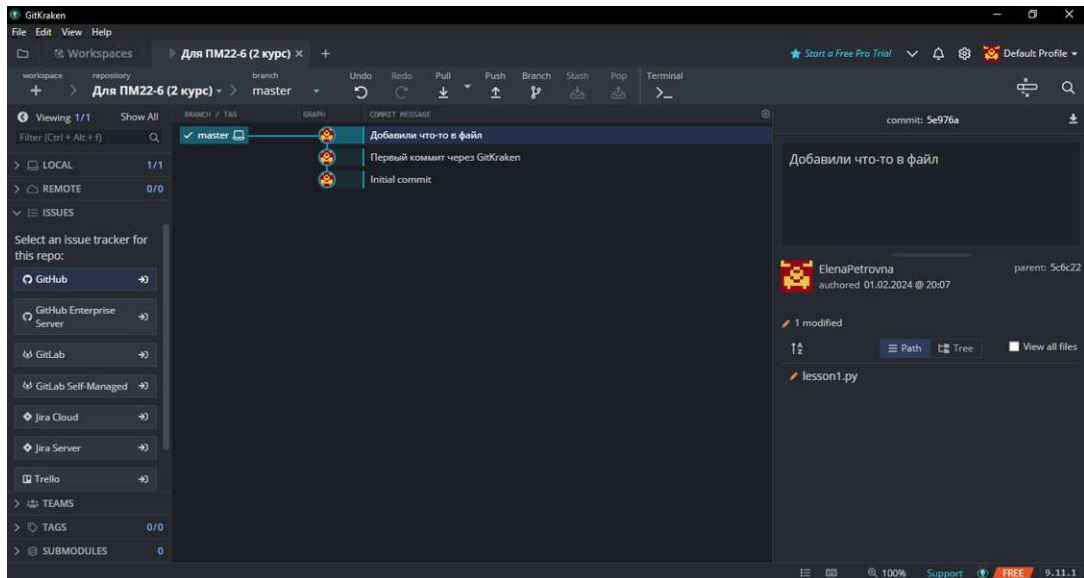
1. Выберите тематику программы, которую собираетесь написать. Создайте для нее рабочую директорию
2. Инициализируйте в рабочей директории репозиторий.



3. Создайте файл для исходного текста программы.



4. Добавьте созданный файл под версионный контроль.
5. Сделайте начальный коммит.
6. Сделайте еще несколько коммитов.



7. Добавьте еще несколько файлов с исходным текстом программы.
8. Создайте новую тематическую ветку.
9. Сделайте несколько коммитов в основную и тематическую ветки.
10. Слейте изменения в основную ветку. Если произошел конфликт слияния, разрешите его и завершите слияние.
11. При получении в процессе разработки программы в стабильно работающем состоянии, слейте это состояние в основную ветку и добавьте к коммиту слияния пометку с номером релиза.

Упражнение 3. Работа с удаленными репозиториями и GitHub.

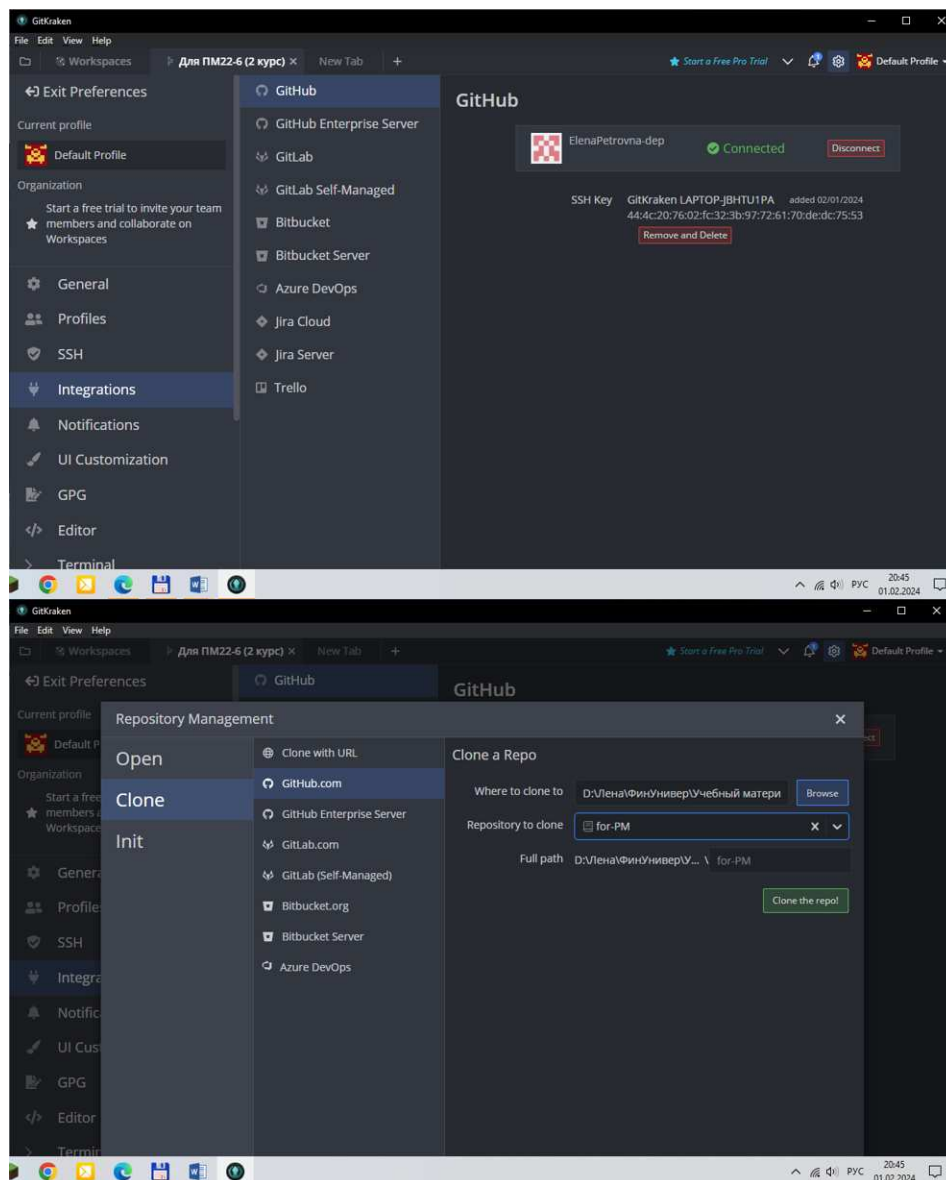
1. Зарегистрироваться на сайте github.com и создать новый пустой репозиторий
2. Создать в этом репозитории файл README

Если вы работаете с git:

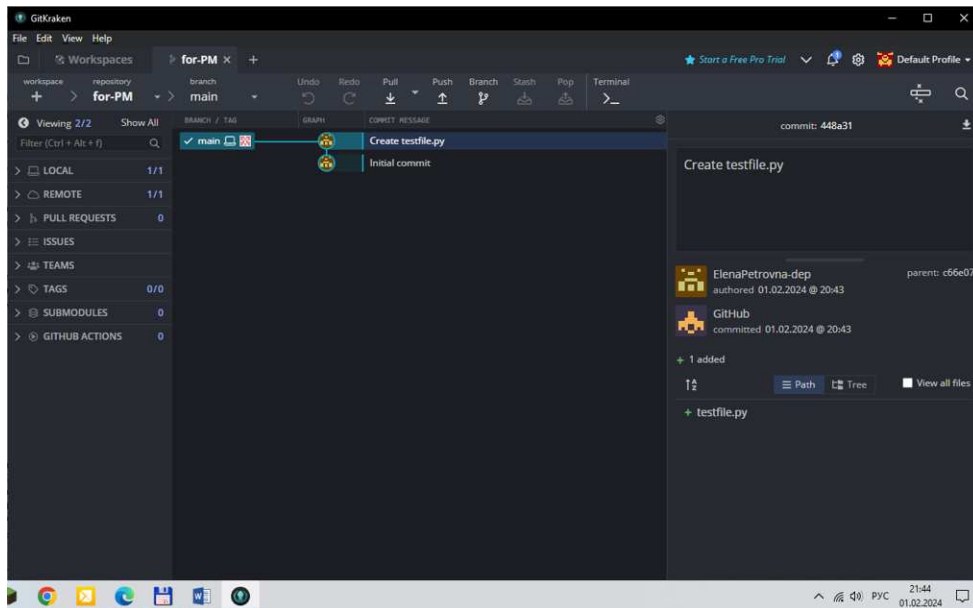
- Склонировать созданный удаленный репозиторий в директорию `~/git/test`
 - На локальной машине пишем скрипт `~/git/test/backup.sh`, с произвольным содержанием
 - Фиксируем скрипт в репозитории (делаем коммит)
 - Обновляем удаленный репозиторий (делаем пуш)

Если вы работаете с gitKraken:

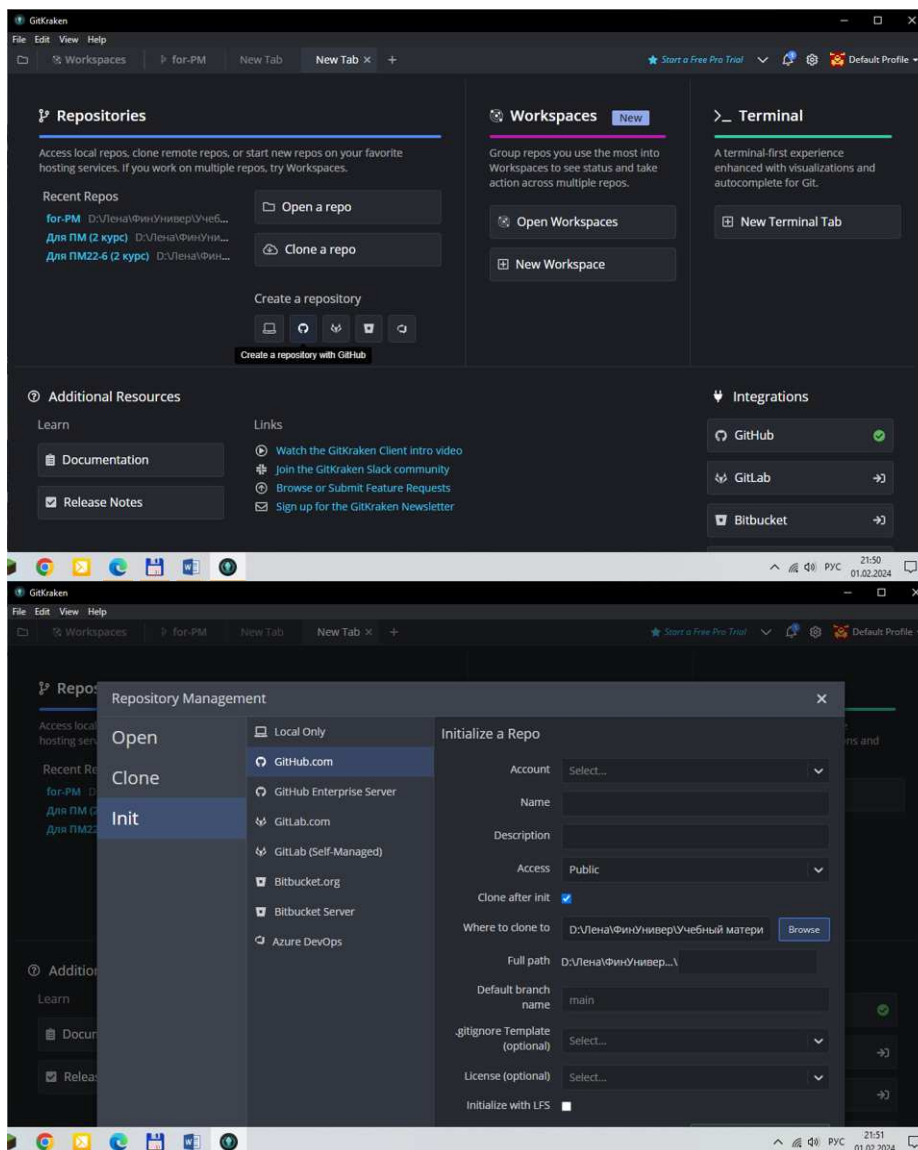
– Склонировать созданный удаленный репозиторий

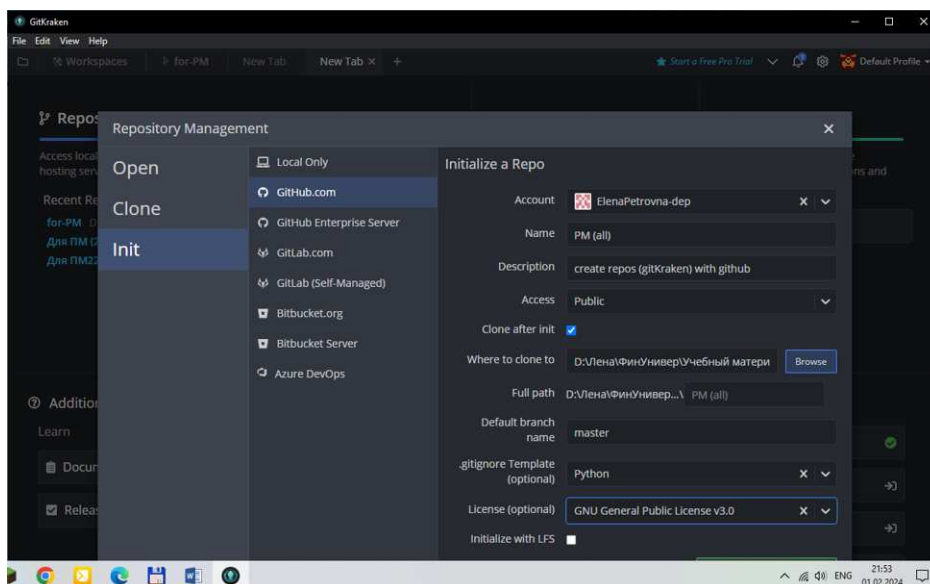


– Фиксируем скрипт в репозитории (делаем коммит)



– Обновляем удаленный репозиторий (делаем пуш)





Задание 12

Лабораторная работа «основы системного программирования»

Цель задания Целью данного задания является развитие навыков системного программирования на языке Python через выполнение серии упражнений, охватывающих работу с файловой системой, процессами, сетями, системной информацией, многопоточностью и асинхронным программированием. Студенты должны не только выполнить указанные задачи, но и глубоко разобраться в используемых библиотеках и конструкциях языка, а также подготовить отчет, описывающий логику работы каждого блока кода.

Задачи для выполнения

- Работа с файловой системой
 - Создание директории по указанному пути.
 - Перечисление файлов в директории и ее поддиректориях.
- Работа с процессами
 - Запуск внешних команд и программ (ping, ls/dir).
 - Анализ результатов выполнения команд.
- Работа с сетями
 - Отправка HTTP-запросов и анализ ответов.

- Установка соединения с сервером через сокеты и обмен сообщениями.
- Работа с системной информацией
 - Отображение информации о системе.
 - Работа с рабочим каталогом и переменными окружения.
- Многопоточность и асинхронное программирование
 - Реализация многопоточности для выполнения длительных задач.
 - Асинхронное выполнение и ожидание результатов операций.

Требования к отчету

Отчет должен содержать:

- Описание каждого упражнения и его цели.
- Пошаговое объяснение реализации задачи, включая описание используемых функций и методов.
- Анализ полученных результатов и выводы.
- Любые трудности, с которыми студенты столкнулись при выполнении задания, и способы их решения.
- Размышления о том, как полученные знания могут быть применены в практических проектах.

Упражнение 1. Работа с файловой системой

- Написать скрипт для создания новой директории. Пусть скрипт принимает имя директории от пользователя и создает ее в указанном месте. Если директория уже существует, скрипт должен сообщить об этом.
- Написать функцию для перечисления всех файлов в данной директории и всех ее поддиректориях. Функция должна выводить абсолютные пути файлов.

Шаги

1. Импортировать модуль `os`. Этот модуль содержит функции для работы с операционной системой, включая файловую систему.

```
import os
```

2. Получить от пользователя путь для создания директории. Используйте функцию `input()` для запроса пути.

```
directory = input("Введите путь для создания директории: ")
```

3. Проверить, существует ли уже такая директория. Используйте `os.path.exists()` для проверки существования директории.

```
if os.path.exists(directory):  
    print("Директория уже существует")  
else:  
    os.makedirs(directory)  
    print(f"Директория {directory} создана")
```

4. Используйте функцию `os.walk()` для обхода директории и всех её поддиректорий. Эта функция генерирует имена файлов в дереве каталогов, обходя дерево сверху вниз или снизу вверх.

```
def list_files(directory):  
    for root, dirs, files in os.walk(directory):  
        for file in files:  
            print(os.path.join(root, file))
```

5. Вызовите функцию и передайте ей путь к директории.

```
directory = input("Введите путь к директории для перечисления файлов: ")  
list_files(directory)
```

Упражнение 2. Работа с процессами

- Написать скрипт для запуска внешних команд или программ. Пусть скрипт запускает команду `ping` для проверки доступности какого-либо ресурса в сети, например, `google.com`. Результат выполнения команды должен быть выведен пользователю.
 - Исследовать возможности модуля `subprocess`. Написать скрипт, который выполняет команду `ls` (для Unix/Linux) или `dir` (для Windows) и обрабатывает результат: выводит количество файлов в директории.
-

1. Импортировать модуль `subprocess`. Этот модуль позволяет запускать новые приложения и команды, подключаться к их каналам ввода/вывода/ошибок и получать их коды возврата.
-

```
import subprocess
```

2. Запустить команду `ping`. Используйте `subprocess.run()` для запуска команды и вывода результата.
-

```
subprocess.run(["ping", "-n", "4", "google.com"])
```

3. Запустить команду `ls` или `dir`. Используйте `subprocess.check_output()` для выполнения команды и получения вывода.
-

```
try:
```

```
output = subprocess.check_output("dir", shell=True, text=True, encoding='cp866')
```

```
print(output)
```

```
files = output.strip().split('\n')
```

```
print(f"Количество элементов: {len(files)}")
```

```
except subprocess.CalledProcessError as e:
```

```
print(f"Ошибка выполнения команды: {e}")
```

4. Подсчитайте количество файлов в выводе, разделив строку по символу перевода строки и посчитав элементы списка.

```
files = output.strip().split('\n')
print(f"Количество файлов: {len(files)}")
```

Упражнение 3. Работа с сетями

- Написать скрипт для проверки доступности веб-сайта. Скрипт должен отправлять HTTP-запрос к выбранному веб-сайту и выводить статус-код ответа.
 - Изучить модуль `socket`. Написать скрипт, который подключается к серверу по заданному адресу и порту, отправляет простое текстовое сообщение и получает ответ.
-

1. Импортировать модуль `requests`. Этот модуль позволяет отправлять HTTP-запросы.
-

```
import requests
```

2. Отправить GET-запрос к веб-сайту. Используйте `requests.get()` и обработайте объект ответа для вывода статус-кода.
-

```
response = requests.get("http://www.google.com")
print(f"Статус-код: {response.status_code}")
```

3. Расширенная проверка доступности веб-сайта с использованием `requests`. Импортируйте необходимых модулей.
-

```
import requests
import time
```

4. Отправка GET-запроса, измерение времени выполнения и получения запроса.

```
start_time = time.time()
response = requests.get("http://www.google.com")
end_time = time.time()
```

5. Анализ и вывод результатов.

```
print(f"Статус-код: {response.status_code}")
print(f"Тип содержимого: {response.headers.get('Content-Type')}")
print(f"Размер контента: {response.headers.get('Content-Length')} байт")
print(f"Время ответа: {end_time - start_time} секунд")
```

6. Работа с сокетами для отправки HTTP-запроса. Импорт модуля socket.

```
import socket
```

7. Определение хоста и порта.

```
host = 'google.com'
port = 80
```

8. Формирование HTTP-запроса.

```
request = f"GET / HTTP/1.1\r\nHost: {host}\r\nConnection: close\r\n\r\n"
```

9. Создание сокета и подключение. Отправка запроса и получение ответа.

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((host, port))

    s.sendall(request.encode())

    response = ""
    while True:
        part = s.recv(4096).decode()
        if not part:
            break
        response += part
```

10. Анализ и вывод ответа.

```
headers, _, body = response.partition('\r\n\r\n')
status_line = headers.splitlines()[0]
status_code = status_line.split(' ')[1]

print(f"Статус-код: {status_code}")
print("Заголовки ответа:")
print('\n'.join(headers.splitlines()[1:]))
```

Упражнение 4. Работа с системной информацией

- Написать скрипт для отображения информации о системе. Скрипт должен выводить информацию о текущей операционной системе, версии Python, а также объем свободной и занятой памяти на диске.
 - Исследование модуля os. Написать функции для получения и вывода текущего рабочего каталога, изменения рабочего каталога и вывода списка всех переменных окружения.
-

1. Импортировать модули os, platform и psutil. Эти модули помогут получить информацию о системе.
-

```
import os, platform, psutil
```

2. Вывести информацию о системе. Используйте функции этих модулей для получения и вывода информации.
-

```
print(f"ОС: {platform.system()} {platform.release()}")
print(f"Версия Python: {platform.python_version()}")
print(f"Свободно на диске: {psutil.disk_usage('/').free / (1024**3):.2f} GB")
```

3. Получить и вывести текущий рабочий каталог.
-

```
print(f"Текущий рабочий каталог: {os.getcwd()}")
```

4. Изменить рабочий каталог и вывести его снова.

```
os.chdir(r'C:\Users\Vit\My\ПРАКТИКУМ 4\dir')  
print(f'Новый рабочий каталог: {os.getcwd()}')
```

5. Вывести список всех переменных окружения.

```
print("Переменные окружения:")  
for key, value in os.environ.items():  
    print(f'{key}: {value}')
```

Упражнение 5. Многопоточность и асинхронное программирование

- Пример многопоточного выполнения задач. Написать скрипт, который запускает несколько потоков для выполнения функции, которая симулирует длительную задачу (например, ожидание с использованием `time.sleep()`).
 - Асинхронное программирование с использованием `asyncio`. Написать асинхронный скрипт для выполнения и ожидания результатов нескольких асинхронных операций (например, асинхронных HTTP-запросов).
-

1. Импортировать модуль `threading` и `time`.

```
import threading, time
```

2. Определить функцию, которая симулирует длительную задачу.

```
def task(name):  
    print(f'Задача {name} началась\n')  
    time.sleep(2)  
    print(f'\nЗадача {name} завершена')
```

3. Запустить несколько потоков для выполнения этой функции.

```
for i in range(5):  
    t = threading.Thread(target=task, args=(f'Поток {i+1}'),)  
    t.start()
```

4. Импортировать модуль `asyncio`.

```
import asyncio
```

5. Определить асинхронную функцию, которая выполняет асинхронную операцию.

```
async def async_task(name):  
    print(f'Асинхронная задача {name} началась')  
    await asyncio.sleep(2)  
    print(f'Асинхронная задача {name} завершена')
```

6. Создать и запустить задачи

```
tasks = [asyncio.create_task(async_task(f'Задача {i+1}')) for i in range(5)]
```

7. Другой способ. Определить и запустить основную асинхронную функцию `main`.

```
async def main():  
    await asyncio.gather(*(async_task(f'Задача {i+1}') for i in range(5)))  
  
await main()
```

Упражнение 6. Загруженность ребра сетевой структуры

Реализуйте класс для расчета загруженности ребра сетевой структуры. Создайте свойство `calculate` для расчета загруженности с возможными значениями от 0 до 100%. Расчет загруженности ребра происходит по формуле:

$$\Delta = \frac{speed_p - speed_f}{speed_p} \times 100\%,$$

где $speed_p$ – плановая скорость ребра, $speed_f$ – фактическая скорость ребра.

Пример расчета для плановой скорости – 110 и фактической скорости – 45:

$$\Delta = \frac{110 - 45}{110} \times 100\% = 59\%$$

Пример расчета для плановой скорости – 40 и фактической скорости – 61:

$$\Delta = \frac{40 - 61}{40} \times 100\% = -52.5\% \rightarrow 0\%$$

Код на python:

class NetworkCongestion:

```
def __init__(
    self,
    plan_speed: float = 30.0,
    fact_speed: float = 60.0,
) -> None:
    self._plan_speed = plan_speed
    self._fact_speed = fact_speed
```

@property

```
def calculate(self) -> float | None:
    ratio = (self._plan_speed - self._fact_speed) / self._plan_speed
    if ratio < 0:
        return 0

    return ratio
```

Критерии балльной оценки различных форм текущего контроля успеваемости содержатся в соответствующих методических рекомендациях Кафедры анализа данных и машинного обучения Факультета информационных технологий и анализа больших данных.

7. Фонд оценочных средств для проведения промежуточной аттестации обучающихся по дисциплине

Перечень компетенций с указанием индикаторов их достижения в процессе освоения образовательной программы содержится в разделе 2. «Перечень планируемых результатов освоения образовательной программы (перечень компетенций) с указанием индикаторов их достижения и планируемых результатов обучения по дисциплине».

**Типовые контрольные задания или иные материалы, необходимые для
оценки индикаторов достижения компетенций, умений и знаний**

Наименование компетенции	Наименование индикаторов достижения компетенции	Результаты обучения (умения и знания), соотнесенные с индикаторами достижения компетенции	Типовые контрольные задания
Способность осуществлять поиск, критически анализировать, обобщать и систематизировать информацию, использовать системный подход для решения поставленных задач (УК-10)	1. Четко описывает состав и структуру требуемых данных и информации, грамотно реализует процессы их сбора, обработки и интерпретации.	<p>Знать: состав и структуру требуемых данных и информации.</p> <p>Уметь: грамотно реализовать процессы сбора, обработки и интерпретации данных и информации.</p>	<p>Выполните обработку текстового файла, в котором зафиксированы факты продаж, необходимого для подсчета суммарных продаж по различным географическим подразделениям фирмы. На основе полученных данных составьте CSV файл, хранящий полученные результаты.</p> <p>Опишите структуру CSV файла фиксирующего факты продаж, необходимого для подсчета суммарных продаж по различным географическим подразделениям фирмы.</p>
	2. Обосновывает сущность происходящего, выявляет закономерности, понимает природу вариативности.	<p>Знать: закономерности реализуемых процессов и природу их вариативности.</p> <p>Уметь: определять закономерности изучаемых процессов и данных, выявлять природу их вариативности.</p>	<p>Опишите закономерности/вариативность в синтаксических конструкциях Python для работы со списками и словарями.</p> <p>Определите количество элементов списка, значения которых совпадают с ключами заданного словаря и количество элементов списка, значения которых равны значениям в словаре.</p>
	3. Формулирует признак классификации, выделяет соответствующие ему группы однородных «объектов», идентифицирует общие свойства элементов этих групп, оценивает полноту результатов	<p>Знать: особенности признаков классификации и оценки результатов классификации.</p> <p>Уметь: разрабатывать программный код, выполняющий классификацию по обозначен-</p>	<p>Опишите подходы используемые в Python для группировки функций, ориентированных на обработку однотипных объектов и решение близких задач.</p> <p>Выполните классификацию с использованием бинарного дерева. Оцените количество операций и глубину дерева</p>

	классификации, показывает прикладное назначение классификационных групп.	ному признаку, ориентироваться в существующем коде, оценивать полноту результатов классификации.	
	4. Грамотно, логично, аргументировано формирует собственные суждения и оценки. Отличает факты от мнений, интерпретаций, оценок и т.д. в рассуждениях других участников деятельности.	Знать: основы логических рассуждений, аргументации, оценки своих суждений и рассуждений других участников. Уметь: проводить сравнительный анализ мнений, интерпретаций, оценок и т.д. в рассуждениях других участников деятельности.	Сформируйте собственные суждения о применимости Python для решения задач машинного обучения. Выполните задачу построения бинарного дерева двумя способами: на основе списков и на основе узлов и ссылок. Оцените эффективность работы каждого способа.
	5. Аргументированно и логично представляет свою точку зрения посредством и на основе системного описания.	Знать: основы аргументированного и логического рассуждения. Уметь: проводить системное описание своей точки зрения, представлять ее аргументированно и логично	Сформулируйте задачи по обработке массивов, которые вы сможете решить на основе языка Python Выскажите свою точку зрения о применимости Python для создания крупной корпоративной транзакционной системы на языке программирования Python.
Способность разрабатывать алгоритмы и программы с использованием современных технологий программирования (ПКН-2)	1. Владеет объектно-ориентированным языком программирования на уровне знания синтаксиса и семантики, основ стандартной библиотеки.	Знать: объектно-ориентированный язык программирования Python на уровне знания синтаксиса и семантики, основ стандартной библиотеки. Уметь: определять на уровне знания синтаксиса и семантику, стандартные библиотеки языка Python, необходимые для решения прикладных задач.	Напишите программу на Python, обрабатывающую текстовую строку так, чтобы в нем все заглавные буквы преобразовать в строчные. Напишите скрипт на Python обрабатывающий текстовый файл и получение на его основе словаря, ключами являются слова текста, а значениями количество встречаемости слов. Сохраните в файле CSV.

	<p>2.Использует инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки).</p>	<p>Знать: инструментальные средства программирования (IDE, SDK, API, популярные фреймворки и библиотеки).</p> <p>Уметь: разрабатывать программы решения задач с использованием инструментальных средств программирования (IDE, SDK, API, популярных фреймворков и библиотек).</p>	<p>Используя программу Jupyter Notebook составьте код на Python, определяющий количество точек на плоскости, находящихся на заданном расстоянии от начала координат. Используйте библиотеку math.</p> <p>Выберите библиотеку Python для работы с массивами. Выполните программный код бинарного поиска данных.</p>
	<p>3.Организовывает кодовую базу, ориентируется в существующем коде, демонстрирует знание общепринятых соглашений и политик в области оформления кода.</p>	<p>Знать: особенности создания программного кода.</p> <p>Уметь: разрабатывать программный код, ориентироваться в существующем коде, применять знание общепринятых соглашений и политик в области оформления кода.</p>	<p>Выберите библиотеку Python для создания и обработки деревьев данных.</p> <p>Выполните программный код вычисления арифметического выражения с помощью двоичного дерева.</p>
	<p>4.Проектирует текстовый, программный или графический интерфейс программной системы исходя из ее назначения.</p>	<p>Знать: основы проектирования различных видов интерфейса программной системы.</p> <p>Уметь: разрабатывать текстовый, программный или графический интерфейс программной системы исходя из ее назначения.</p>	<p>Реализуйте программный код вычисления различных арифметических операций, выбор которых осуществляется с помощью разработанного меню.</p> <p>Реализуйте программу на Python которой в качестве аргумента командной строки передается имя CSV-файла, в первом столбце находятся числа, которые необходимо отсортировать. Программа создает новый файл, в котором первый столбец отсортирован.</p>

Примерные вопросы для подготовки к зачетам

1 семестр

1. Функции модуля math.
2. Инструкция if...else.
3. Инструкция цикла while.
4. Инструкция цикла for.
5. Функция range.
6. Строки. Операции над строками (+, *, in, доступ по индексу [], получение среза). Функция len.
7. Методы строк: strip, find, count, replace.
8. Списки в Python. Создание: создание пустого списка, методы append, split, функция list. Генераторы списков.
9. Создание копии списка (срезы, функции list и deepcopy).
10. Основные операции над списками (+, *, in, доступ по индексу [], получение среза). Перебор элементов списка.
11. Добавление и удаление элементов списка (методы append, insert, pop, remove). Методы reverse, join. Функция map.
12. Сортировка списков. Параметры метода sort: key, reverse.
13. Кортежи. Создание. Создание пустого кортежа и кортежа из одного элемента. Операции (+, *, in, доступ по индексу [], получение среза).
14. Функции tuple, len.
15. Распаковка последовательности.
16. Словари. Создание словаря. Функции dict, zip.
17. Операции над словарями ([], in, del). Функция len.
18. Перебор элементов словаря. Методы get, clear, copy, keys, values.
19. Множества. Создание. Функции set, len.
20. Операции над множествами: in, |, &, -, ^, <=, >=, <, >, ==. Методы add, discard.
21. Создание и вызов функции.
22. Передача аргументов в функцию. Необязательные параметры функций. Функции в качестве аргументов.
23. Глобальные и локальные переменные.
24. Анонимные функции.
25. Создание и использование модулей. Инструкции import и from.
26. Пакеты. Использование пакетов.

2 семестр

1. Инструкция try ... except ... else ... finally.
2. Инструкции raise и assert.
3. Инструкция with ... as.
4. Классы встроенных исключений.
5. Работа с текстовыми файлами. Открытие файла (функция open) и закрытие файла (метод close). Чтение текстового файла (методы read, readline, readlines). Перебор строк файла в цикле for. Запись в текстовый файл (метод write, функция print с параметром file).
6. Сохранение объектов в файл (функции dump и load модуля pickle).
7. Понятие класса и объекта. Определение класса и создание экземпляра класса.
8. Методы класса. Параметр self. Статические методы. Закрытые методы.
9. Атрибуты класса и экземпляра класса. Доступ к атрибуту. Закрытые атрибуты.
10. Свойства. Создание и использование свойства.
11. Наследование. Базовый и производный классы. Переопределение методов.
12. Специальные методы. Перегрузка операторов.
13. Функции map, filter, reduce, any, all.
14. Декораторы функций.
15. Итераторы.
16. Функции-генераторы.
17. Массивы. Использование массивов.
18. Стеки. Использование стеков. Реализация стека.
19. Очереди. Использование очереди. Реализация очереди.
20. Связные списки. Использование связных списков. Реализация связных списков.
21. Бинарные деревья. Использование бинарных деревьев. Реализация бинарных деревьев.
22. Бинарный поиск.
23. Обменные сортировки.
24. Сортировка Шелла.
25. Быстрая сортировка.

3 семестр

1. Основные принципы объектно-ориентированного проектирования.
2. Шаблон проектирования: интерфейс
3. Шаблон проектирования: делегирование.
4. Шаблон проектирования: фабричный метод
5. Шаблон проектирования: абстрактная фабрика
6. Шаблон проектирования: строитель
7. Шаблон проектирования: адаптер
8. Шаблон проектирования: мост
9. Шаблон проектирования: декоратор
10. Шаблон проектирования: фасад
11. Шаблон проектирования: цепочка обязанностей
12. Шаблон проектирования: команда
13. Шаблон проектирования: посредник
14. Шаблон проектирования: наблюдатель
15. Шаблон проектирования: состояние
16. Шаблон проектирования: стратегия
17. Шаблон проектирования: посетитель
18. Шаблон проектирования: шаблонный метод.
19. Событийно-ориентированное программирование.
20. События и обработчики событий. Виды событий.
21. События мыши и клавиатуры.
22. tkinter, PyQt, PyGTK – особенности и отличия.
23. Главный цикл программы.
24. Основные элементы управления: кнопки, ползунки, поля ввода.
25. Работа с графикой.
26. Работа с путями в Windows и Linux.
27. Основные форматы хранения данных: CSV, XML, JSON.

4 семестр

1. Получение и разбор HTML-страниц.
2. Библиотеки HTML-парсинга.
3. Сокеты.
4. Клиент-серверные приложения.
5. Обращение к внешним API.
6. Многопоточность.
7. Загруженность ребра сетевой структуры.

8. Библиотеки многопоточности и многопроцессности.
9. Отправка и получение электронных писем.
10. Основные виды тестирования программного обеспечения.
11. Модульное и интеграционное тестирование.
12. Понятие регрессии и регрессионного тестирования.
13. Библиотеки автоматизированного тестирования.
14. Разработка через тестирование.

Примерные задания для подготовки к зачетам

1 семестр

1. Чему будет равно значение переменной A после выполнения оператора:
`A = [i+10 for i in range(5,0,-1)]`
2. Имеется строка `S="1234567890"`. Чему равно значение S1, если
`S1 = S[1:2] + S[4: :-1]`
4. Используя генератор словарей, для заданного натурального числа n создайте словарь D, в котором будет ровно n элементов. Элементами словаря являются `{'s1':10, 's2': 20, 's3': 30, ...}`, т.е. значение равно номеру элемента (нумерация с 1), умноженному на 10, а ключ состоит из символа 's', к которому дописан номер.

2 семестр

1. Имеется список L вида `[[1, 3, 65], [11, 5, 6], [15, 33, 11]]`. Отсортируйте список по возрастанию последнего значения элемента.
2. После выполнения приведенного кода будет напечатано ...

```
def data(d=[]):  
    d.append(1)  
    return d
```

```
a=data()  
b=data()  
c=data([3])  
print(b,c)
```

- a) `[1] [3]`
- b) `[2] [4]`

- c) [1,1] [3,1]
- d) [1,2] [3,4]
- e) Будет выведено сообщение об ошибке

3. После выполнения приведенного кода будет напечатано ...

```
class Class1:  
    def __init__(self, n):  
        self._x = n  
  
    def gx(self):  
        return self._x+2  
  
    x = property(gx)
```

```
c=Class1(2)  
print(c.x)
```

- a) 2
- b) 4
- c) None
- d) Возникнет ошибка

3 семестр

1. Приведите пример приватного IP-адреса

- a. 196.168.0.1
- b. 127.0.0.10
- c. 87.250.250.242
- d. 173.194.73.113

2. Приведите пример публичного IP-адреса

- a. 8.8.8.8
- b. 127.0.0.1
- c. 196.168.0.1
- d. 10.38.51.16

3. Соглашение о порядке и способе связи между компьютерами это:

- a. сетевой протокол
- b. сетевой интерфейс
- c. коммутация
- d. адресация

4. Как называется самый распространенный стандарт на архитектуру локальных сетей на основе кабельного соединения

- a. Ethernet
- b. Wi-Fi
- c. TCP
- d. GlobalNet X

4 семестр

1. Приведите пример стандарта из серии IEEE 802

- a. Wi-Fi
- b. витая пара
- c. POSIX
- d. USB

2. Назовите команду, отображающую основную информацию о сетевых подключениях в ОС Linux

- a. ipconfig
- b. ifconfig
- c. netstat
- d. traceroute

3. Назовите команду, отображающую основную информацию о сетевых подключениях в ОС Windows

- a. ifconfig
- b. ipconfig
- c. netstat
- d. tracert

4. Назовите пример сетевой топологии

- a. звезда
- b. снежинка

с. лабиринт
d.круговая

8. Перечень основной и дополнительной учебной литературы, необходимой для освоения дисциплины

Основная литература:

1. Гуриков, С. Р. Основы алгоритмизации и программирования на Python: учебное пособие / С. Р. Гуриков. — Москва : ИНФРА-М, 2022. — 343 с. — (Высшее образование: Бакалавриат). — ЭБС ZNANIUM. - URL: <https://znanium.com/catalog/product/1356003> (дата обращения: 06.03.2024). — Текст : электронный.
2. Шелудько, В. М. Язык программирования высокого уровня Python. Функции, структуры данных, дополнительные модули : учебное пособие / В. М. Шелудько; Южный федеральный университет. - Ростов-на Дону; Таганрог: Издательство Южного федерального университета, 2017. - 107 с. — ЭБС ZNANIUM. - URL: <https://znanium.com/catalog/product/1021664> (дата обращения: 06.03.2024). — Текст : электронный.
3. Жуков, Р. А. Язык программирования Python: практикум: учебное пособие / Р. А. Жуков. — Москва : ИНФРА-М, 2022. — 216 с. — (Высшее образование: Бакалавриат). — ЭБС ZNANIUM. — URL: <https://znanium.com/catalog/product/1689648> (дата обращения: 06.03.2024). — Текст : электронный.
4. Практикум по программированию на языке Python: учебное пособие для студентов, обучающихся по направлениям "Прикладная математика и информатика", "Прикладная информатика", "Информационная безопасность", "Бизнес-информатика" (программа подготовки бакалавра) / Р. И. Горохова, Е. П. Догадина, В. И. Долгов, В. И. Макрушин; Финуниверситет,

Департамент анализа данных и машинного обучения факультета информационных технологий и анализа больших данных. — Москва : Финуниверситет, 2023. — ЭБ Финуниверситета. — URL: <http://elib.fa.ru/fbook/books137296.pdf> (дата обращения). — Текст : электронный.

Дополнительная литература:

1. Северенс, Ч. Введение в программирование на Python / Ч. Северенс. — 2-е изд., испр. — Москва : Национальный Открытый Университет «ИНТУИТ», 2016. — 231 с. — ЭБС Университетская библиотека online. — URL: <https://biblioclub.ru/index.php?page=book&id=429184> (дата обращения: 06.03.2024). — Текст : электронный.
2. Дроздов, С. Н. Структуры и алгоритмы обработки данных: учебное пособие / С. Н. Дроздов. — Таганрог : Южный федеральный университет, 2016. — 228 с. — ЭБС ZNANIUM. — URL: <https://znanium.com/catalog/product/991928> (дата обращения: 06.03.2024). — Текст : электронный.

9. Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины

1. Pyru 1.0.9 [Электронный ресурс]: сайт. — Режим доступа: <https://pypi.python.org/pypi/pyru>.
2. Python Data Analysis Library [Электронный ресурс]: сайт. — Режим доступа: <http://pandas.pydata.org/>.
3. Python Documentation [Электронный ресурс]: сайт. — Режим доступа: <http://python.org/doc/>.
4. Python Standard Library [Электронный ресурс]: сайт. — Режим доступа: <https://docs.python.org/2/library/>.
5. Scikit-learn Machine Learning in Python [Электронный ресурс]: сайт. — Режим доступа: <http://scikit-learn.org>.

6. Официальный сайт продукта <https://www.python.org/>.
7. Портал Финансового университета <http://www.fa.ru/>.
8. Каталог курсов Интернет Университета Информационных Технологий <http://www.intuit.ru/>.
9. The Python Tutorial // <https://docs.python.org/3/tutorial/index.html>.
10. The Python Standard Library <https://docs.python.org/3/library/index.html>.
11. SciPy // <http://docs.scipy.org/doc/scipy/reference/>.
12. NumPy User Guide // <http://docs.scipy.org/doc/numpy/user/index.html>.
13. Электронная библиотека Финансового университета (ЭБ) <http://elib.fa.ru/>.
14. Электронно-библиотечная система BOOK.RU <http://www.book.ru>.
15. Электронно-библиотечная система «Университетская библиотека ОН-ЛАЙН» <http://biblioclub.ru/>.
16. Электронно-библиотечная система Znanium <http://www.znaniy.com>.
17. Электронно-библиотечная система издательства «ЮРАЙТ» <https://urait.ru/>.
18. Электронно-библиотечная система издательства Проспект <http://ebs.prospekt.org/books>.
19. Электронно-библиотечная система издательства «Лань» <https://e.lanbook.com/>.
20. Электронная библиотека Издательского дома «Гребенников» <https://grebennikon.ru/>.
21. Деловая онлайн-библиотека Alpina Digital <http://lib.alpinadigital.ru/>.
22. Научная электронная библиотека eLibrary.ru <http://elibrary.ru>.
23. Национальная электронная библиотека <http://нэб.рф/>.
24. Финансовая справочная система «Финансовый директор» <http://www.1fd.ru/>.
25. Ресурсы информационно-аналитического агентства по финансовым рынкам Cbonds.ru <https://cbonds.ru/>.
26. СПАРК <https://spark-interfax.ru/>.

27. Бесплатный курс «Основы Python-разработки».

<https://practicum.yandex.ru/python-free/>.

28. Программирование на Python. <https://stepik.org/course/67/promo>.

10. Методические указания для обучающихся по освоению дисциплины

Лекционные занятия проводятся в соответствии с тематическим планом, при изложении материала рекомендуется использовать презентации в среде PowerPoint программный код из Jupyter Notebook и фрагменты печатных материалов по теме лекции.

В ходе интерактивных занятий следует проводить разбор конкретных примеров программного кода из Jupyter Notebook.

11. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине, включая перечень необходимого программного обеспечения и информационных справочных систем

11.1. Комплект лицензионного программного обеспечения:

1. Операционная система. Пакет офисных программ.
2. Антивирус Kaspersky
3. Дистрибутив Python Anaconda (свободно распространяемое ПО).
4. Браузер.
5. Файловый менеджер Far

11.2. Современные профессиональные базы данных и информационные справочные системы

1. Информационно-правовая система «Гарант».
2. Информационно-правовая система «Консультант Плюс».
3. Электронная энциклопедия: <http://ru.wikipedia.org/wiki/Wiki>
4. Система комплексного раскрытия информации «СКРИН» - <http://www.skrin.ru>

11.3. Сертифицированные программные и аппаратные средства защиты информации - не используются.

12. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине

Материально-техническая база Финансового университета, необходимая для осуществления образовательного процесса по данной дисциплине, в соответствии с требованиями ФОС ВО включает в себя специальные помещения для проведения лекций, семинарских занятий, групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации; помещения укомплектованы специализированной мебелью и техническими средствами обучения, необходимыми для представления информации большой аудитории.

Помещения для самостоятельной работы студентов включают в себя библиотеку с читальным залом, укомплектованную в соответствии с существующими нормами необходимой учебной и учебно-методической литературой и иными материалами; медиатеку с выходом в Интернет, компьютерные классы с возможностью работы в Интернет; аудитории для консультационной деятельности.