МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Программирование и обработка данных на Python

Трудоемкость		Семестр	Вид контроля	Контактная работа, час.	Занятий лекц. типа, час.	Лаборат. занятий, час.	Практич. занятий, час.	Конс, час.	СРО, час.
зач. ед.	час.								
4.0	144.0	1	Дифференцированный зачет	105.6	32.0	64.0	0.0	0.0	38.4
4.0	144.0	**ИТОГО**		105.6	32.0	64.0	0.0	0.0	38.4

Санкт-Петербург

2025 г.

РАБОЧАЯ ПРОГРАММА ДИСЦИПЛИНЫ

Разработана: Казанцев Даниил Владимирович

1. ЦЕЛИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью освоения дисциплины является достижение следующих результатов обучения:

Знания	Умения	Навыки
Типы данных, Принципы ООП (инкапсуляция, наследование, полиморфизм, абстракция), Хранение данных, Классы, как создавать и использовать функции и модули, Основы языка программирования Python, Условные выражения и циклы, Принципы работы с пространством имен , Принципы работы с областью видимости переменных, Принципы наследования в Python, Механизмы разрешения имен в ООП, Основы синтаксиса и управляющих структур языка Python, структуры и принципы настройки окружения, Условия, циклы, логические выражения, Основные принципы ООП, синтаксис классов, методы экземпляров и классов	определять и реализовывать собственные классы с атрибутами и методами, применять объекты и классы для решения практических задач, Создание классов, Проектирование иерархий, Установка Python, настройка IDE, Работа со строками, Работа со списками, Работа со словарями, Написание условных конструкций и циклов, Анализ порядка разрешения имен в сложных иерархиях, переопределение специальных методов для тонкой настройки поведения объектов, отладка непредвиденных конфликтов имен, Создание классов-наследников, расширение функционала родительских классов, реализация иерархий для повторного использования кода, применение конструктора (init), организация кода в объектно-ориентированном стиле, настройка среды разработки (PyCharm, VSCode, Jupyter Notebook)	рефакторинга, тестирования и отладки кода, Моделирование объектов, Конфигурация окружения, работа с репозиториями, Эффективное использование структур данных, Оптимизация управления потоком выполнения, Точная настройка логики доступа к атрибутам, Создание расширяемых ООП-архитектур, Применение MRO для улучшения контрольного потока в иерархиях классов, Проектирование объектной модели для решения практических задач, переход от процедурного к объектно-ориентированному подходу, улучшение читаемости кода за счет ООП-парадигм, Четкая структуризация кода с учетом областей видимости, предотвращение труднообнаружимых ошибок, формирование безопасных и понятных пространств имен в масштабных проектах

2. СТРУКТУРА И СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

№ раздела	Наименование раздела дисциплины	Распределение часов по дисциплине, часы						
		Контактная работа	Занятия лекционного типа	Лабораторные занятия	Практические занятия	Консультации	СРО	Всего часов
1 семестр								
1	Введение в Python и настройка рабочего окружения	13.2	4.0	8.0	0.0	0.0	4.0	17.2
2	Неизменяемые и изменяемые типы данных	13.2	4.0	8.0	0.0	0.0	4.0	17.2
3	Управляющие конструкции	13.2	4.0	8.0	0.0	0.0	4.0	17.2
4	Повторное использование кода	13.2	4.0	8.0	0.0	0.0	4.0	17.2
5	Пространство имен и области видимости	13.2	4.0	8.0	0.0	0.0	4.0	17.2

6	ООП. Классы	13.2	4.0	8.0	0.0	0.0	4.0	17.2
7	ООП. Наследование	13.2	4.0	8.0	0.0	0.0	4.0	17.2
8	ООП. Разрешение имен атрибутов и методов	13.2	4.0	8.0	0.0	0.0	10.4	23.6
ИТОГО:		**105.6**	**32.0**	**64.0**	**0.0**	**0.0**	**38.4**	**144.0**

СОДЕРЖАНИЕ РАЗДЕЛОВ

№ раздела	Наименование раздела дисциплины	Содержание
1 семестр		
1	Введение в Python и настройка рабочего окружения	Знакомство с языком
		Настройка рабочего окружения
2	Неизменяемые и изменяемые типы данных	Введение в типы данных: строки, списки, кортежи, словари
		Работа с неизменяемыми и изменяемыми типами данных
3	Управляющие конструкции	Условные выражения и циклы
4	Повторное использование кода	Функции и модули
5	Пространство имен и области видимости	Принципы работы с пространством имен и областью видимости переменных
6	ООП. Классы	Введение в объекты и классы
7	ООП. Наследование	Принципы наследования в Python
8	ООП. Разрешение имен атрибутов и методов	Механизмы разрешения имен в ООП

3. УЧЕБНО-МЕТОДИЧЕСКОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

Список литературы:

1. Никитина Т. П., Королев Л. В. Программирование. Основы Python для инженеров — Издательство "Лань", 2023 — 156 с. — ISBN 978-5-507-45284-2 — Текст : электронный // ЭБС Лань — URL: https://e.lanbook.com/book/302720
2. Саммерфилд М. Python на практике — Издательство "ДМК Пресс", 2014 — 338 с. — ISBN 978-5-97060-095-5 — Текст : электронный // ЭБС Лань — URL: https://e.lanbook.com/book/66480

Иные ресурсы:

1. Реализуем проекты на Python. Практическое руководство - URL: https://realpython.com/
2. Python Software Foundation. Официальная документация Python. - URL: https://docs.python.org/
3. Хиллман Ч. Python. Разработка приложений. - М.: Альпина Паблишер, 2021.
4. МакКинни У. Python и анализ данных. - М.: ДМК Пресс, 2020.
5. Рамальо Л. Python. Карманный справочник. - СПб.: Питер, 2021. Шоу З. А. Изучаем Python. Часть 1. Основы программирования. - СПб.: Питер, 2019.
6. Блажко В. И. Python: от новичка до профессионала. - М.: Наука и техника, 2020.
7. Гриффитс Д., Гриффитс П. Изучаем Python. Простая наука программирования. - СПб.: Питер, 2021.
8. Саммерфилд М. Программирование на Python 3. Подробное руководство. - М.: БХВ-Петербург, 2020.
9. Бейкер Д. Программирование на Python. Полное руководство. - СПб.: Питер, 2021.
10. Лутц М. Изучаем Python. 5-е издание. - М.: Вильямс, 2019. Свейгарт А. Автоматизация рутинных задач с помощью Python. - М.: Питер, 2020.

4. ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ПРОВЕДЕНИЯ ТЕКУЩЕЙ АТТЕСТАЦИИ ПО ДИСЦИПЛИНЕ

Порядок оценки освоения обучающимися учебного материала определяется содержанием следующих разделов дисциплины:

№ п/п	Наименование раздела дисциплины (модуля)	Оценочные средства контроля успеваемости	Тип оценочного средства
1 семестр			
1	Введение в Python и настройка рабочего окружения	Решение шифров Цезаря и Виженера	Лабораторная работа
2	Неизменяемые и изменяемые типы данных	Решение шифров Цезаря и Виженера	Лабораторная работа
3	Управляющие конструкции	Алгоритм решения игры "Сапёр"	Лабораторная работа
4	Повторное использование кода	Алгоритм решения игры "Сапёр"	Лабораторная работа
5	Пространство имен и области видимости	Реализация системы контроля версий	Лабораторная работа
6	ООП. Классы	Реализация системы контроля версий	Лабораторная работа
7	ООП. Наследование	Работа с внешним API (Телеграм)	Лабораторная работа
8	ООП. Разрешение имен атрибутов и методов	Работа с внешним API (Телеграм)	Лабораторная работа

5. ТИПОВЫЕ КОНТРОЛЬНЫЕ ЗАДАНИЯ ИЛИ ИНЫЕ МАТЕРИАЛЫ, НЕОБХОДИМЫЕ ДЛЯ ОЦЕНКИ ДОСТИЖЕНИЯ ЗАПЛАНИРОВАННЫХ РЕЗУЛЬТАТОВ ОБУЧЕНИЯ

№ оценочного средства	Название	Тип	Ключевая точка	Мин. балл	Макс. балл	Оценивает раздел(-ы)
1 семестр						
1	Решение шифров Цезаря и Виженера	Лабораторная работа	**нет**	0	20	Введение в Python и настройка рабочего окружения Неизменяемые и изменяемые типы данных
2	Алгоритм решения игры "Сапёр"	Лабораторная работа	**нет**	0	20	Управляющие конструкции Повторное использование кода
3	Реализация системы контроля версий	Лабораторная работа	**нет**	0	20	Пространство имен и области видимости ООП. Классы
4	Работа с внешним API (Телеграм)	Лабораторная работа	**нет**	0	20	ООП. Наследование ООП. Разрешение имен атрибутов и методов
5	Дифференцированный зачет	Дифференцированный зачет	-	0	20	Все разделы

Оценочные средства 1 семестра

[1] **Решение шифров Цезаря и Виженера**

Лабораторная работа

Описание: ЛР1. Шифрование

Требования к реализации

Все перечисленные ниже функции должны быть реализованы в соответствующих модулях

Модуль caesar.py - шифр Цезаря

Реализуйте функции для шифрования и дешифрования текста с использованием шифра Цезаря.

```
def encrypt(plaintext: str, shift: int) -> str
def decrypt(ciphertext: str, shift: int) -> str
```

Требования к алгоритму:

* Шифрованию подлежат все печатные символы ASCII (с кодами от 32 до 126 включительно).
* Сдвиг должен корректно "заворачиваться" в пределах указанного диапазона символов.
* Функции должны корректно обрабатывать как положительные, так и отрицательные значения сдвига.

Пример: encrypt("Hello, World!", 3) должен вернуть Khoor/#Zruog$

Модуль vigenere.py - шифр Виженера

Реализуйте функции для шифрования и дешифрования текста с использованием полиалфавитного шифра Виженера.

```
def encrypt(plaintext: str, keyword: str) -> str
def decrypt(ciphertext: str, keyword: str) -> str
```

Требования к алгоритму:

* Шифрованию подлежат все печатные символы ASCII (с кодами от 32 до 126).
* Ключевое слово (keyword) применяется циклически.
* Символы в ключевом слове нечувствительны к регистру ('A' и 'a' эквивалентны сдвигу на 0, 'B' и 'b' — на 1, и т.д.).
* Символы, не входящие в латинский алфавит, в ключевом слове должны игнорироваться.

Пример: encrypt("attack at dawn", "LEMON") должен вернуть lx!opv$m#-oe$|

Модуль rsa.py - ассиметричное шифрование RSA

Реализуйте полный цикл асимметричного шифрования RSA, включая генерацию ключей и обработку текстовых сообщений.

Математические основы

```
def is_prime(n: int) -> bool
def gcd(a: int, b: int) -> int
def multiplicative_inverse(e: int, phi:int) -> int
```

Генерация ключей

```
def generate_keypair(p: int, q: int) -> tuple[tuple[int, int], tuple[int, int]]
```

* Принимает два простых числа p и q
* Возвращает пару ключей (открытый, закрытый) в формате ((e, n), (d, n))

Шифрование текста

```
def encrypt(public_key: tuple[int, int], text: str) ->
list[int]
```

* Принимает открытый ключ и строку.
* Возвращает список зашифрованных числовых кодов символов.

```
def decrypt(private_key: tuple[int, int], cipher_list:
list[int])                    ->                    str
```

* Принимает закрытый ключ и список зашифрованных кодов.
* Возвращает исходную строку.

Модуль hack.py - криптоанализ

Реализуйте функцию для автоматического взлома шифра Цезаря, примененного к англоязычному тексту.

```
def hack(ciphertext: str) -> tuple[str, int]
```

Требования к алгоритму:

1) Перебирает все ключи: В цикле попробуйте расшифровать сообщение каждым возможным ключом (от 1 до 25).
2) Оценивает результат: После каждой расшифровки ваша программа должна оценить, насколько полученный текст похож на осмысленный английский текст.
3) Использует частотный анализ: Самый надежный способ оценки — сравнить частоту букв в расшифрованном тексте с эталонной частотой для английского языка (самые частые буквы — e, t, a, o, i, n, ...).
4) Выносит вердикт: Функция должна вернуть наиболее вероятный исходный текст и ключ, который был для этого использован.

Критерии оценки

* 3 балла — Корректная реализация модуля caesar.py в соответствии со всеми требованиями.
* 4 балла — Корректная реализация модуля vigenere.py в соответствии со всеми требованиями.
* 9 баллов — Корректная реализация модуля rsa.py: 3 балла за математические функции (is_prime, gcd, multiplicative_inverse).2 балла за generate_keypair.4 балла за функции encrypt и decrypt для текста.
* 3 балла — Корректная реализация модуля hack.py.
* 1 балл — Качество кода: читаемость, структурированность, наличие документации (docstrings).

[2] **Алгоритм решения игры "Сапёр"**

Лабораторная работа

Описание:

Введение и теоретическая основа

Обзор

В данной лабораторной работе вам предстоит разработать полнофункциональную игру «Сапёр» (Minesweeper) для командной строки и создать автоматизированный решатель, способный её проходить.

Проект охватывает три ключевых этапа:

1) Создание игрового движка: Вы разработаете надежную основу для игры, используя классы и списки для моделирования игрового поля.
2) Реализация детерминированного решателя: Вы напишете модуль, который делает логически безупречные выводы, используя множества и словари для эффективной обработки ограничений.
3) Разработка вероятностного модуля: Вы создадите алгоритм, который принимает обоснованные решения в ситуациях неопределенности, применяя рекурсивный перебор для анализа всех возможных состояний.

“Сапёр” как вычислительная задача

Формализация правил: Игра «Сапёр» представляет собой прямоугольную сетку размером n×m, под некоторыми ячейками которой скрыты мины. Цель игрока — открыть все ячейки, не содержащие мин. При открытии ячейки, под которой нет мины, отображается число от 0 до 8. Это число указывает на точное количество мин, находящихся в восьми соседних ячейках. Если в соседних ячейках мин нет (число 0), то все они открываются автоматически. Игра считается выигранной, когда все безопасные ячейки открыты. Поражение наступает при открытии ячейки с миной.

Вычислительная сложность: Задача определения, является ли данная конфигурация поля «Сапёра» разрешимой, является co-NP-полной. Это означает, что не существует известного эффективного (полиномиального по времени) алгоритма, который мог бы гарантированно решить любую конфигурацию. Этот факт объясняет, почему ваш решатель должен быть многоуровневым: он должен уметь не только делать быстрые логические выводы, но и быть готовым к ситуациям, когда детерминированная логика заходит в тупик, и единственным выходом остаётся сделать обоснованное, но рискованное предположение.

Цели и задачи лабораторной работы

* Разработка эмулятора игры: Создать интерактивное консольное приложение, позволяющее пользователю играть в «Сапёр».
* Реализация детерминированного решателя: Написать модуль, который применяет базовые и продвинутые логические правила для нахождения гарантированно безопасных и заминированных ячеек.

* Применение методов CSP: Смоделировать задачу как проблему удовлетворения ограничений (Constraint Satisfaction Problem, CSP) для решения сложных конфигураций.
* Разработка вероятностного решателя: Реализовать модуль, который вычисляет точные вероятности нахождения мин и делает наиболее безопасное предположение.
* Анализ производительности: Провести эмпирический анализ эффективности разработанного решателя.

Часть I - Эмуляция игры “Сапёр” (6 баллов)

На этом этапе вы создадите ядро игры, уделив особое внимание правильному проектированию структур данных.

Подзадача: Структура данных и управление состоянием (3 балла)

Требования к именованию: Вся логика игрового движка должна находиться в файле minesweeper_engine.py.

* Модель ячейки: Вам необходимо создать класс Cell для хранения состояния каждой ячейки. Для этой цели идеально подходит dataclasses из стандартной библиотеки Python. Класс должен содержать как минимум следующие атрибуты:is_mine: boolis_revealed: boolis_flagged: booladjacent_mines: int (от 0 до 8)
* is_mine: bool
* is_revealed: bool
* is_flagged: bool
* adjacent_mines: int (от 0 до 8)
* Модель игрового поля: Игровое поле должно быть представлено как вложенный список: list[list[Cell]]. Такая структура обеспечивает интуитивно понятный доступ к любой ячейке по её координатам (row, col).
* Генерация поля: Вам следует реализовать функцию generate_board(rows, cols, num_mines, first_click_coords). Важным требованием является обеспечение гарантированно безопасного первого хода. Расстановка мин должна происходить после первого хода игрока, исключая ячейку, на которую был сделан ход, из списка кандидатов для размещения мины.
* Предварительный расчет смежности: После расстановки мин необходимо выполнить однократный проход по всей сетке для вычисления значения adjacent_mines для каждой ячейки. Этот предварительный расчет (пре-кэширование) является важной оптимизацией.

Подзадача: Интерфейс командной строки и игровая логика (3 балла)

Требования к именованию: Основной исполняемый файл для взаимодействия с пользователем должен называться play_minesweeper.py.

* Разработка интерфейса командной строки (CLI): После каждого хода программа должна выводить на экран актуальное состояние игрового поля. Используйте следующие символы:# — неоткрытая ячейкаF — флаг* — мина (после проигрыша)0-8 — открытая безопасная ячейка
* # — неоткрытая ячейка

* F — флаг
* * — мина (после проигрыша)
* 0-8 — открытая безопасная ячейка
* Обработка действий пользователя: Программа должна корректно обрабатывать команды в формате:open <row> <col> (или o <row> <col>): Открывает ячейку.flag <row> <col> (или f <row> <col>): Устанавливает/снимает флаг.
* open <row> <col> (или o <row> <col>): Открывает ячейку.
* flag <row> <col> (или f <row> <col>): Устанавливает/снимает флаг.
* Каскадное открытие: Если открытая ячейка имеет 0 соседних мин, все её 8 соседей должны быть автоматически открыты. Рекурсивная реализация этого механизма может привести к ошибке RecursionError. Вам необходимо реализовать итеративный алгоритм, используя очередь (collections.deque) для поиска в ширину (BFS). Это продемонстрирует ваше умение выбирать подходящие структуры данных для решения классических алгоритмических задач.

Определение состояний завершения игры: Реализуйте логику, которая проверяет условия победы (все безопасные ячейки открыты) и поражения (открыта ячейка с миной).

Часть II - Детерминированный решатель (9 баллов)

Требования к именованию: Вся логика решателя должна находиться в файле solver.py в классе Solver

Подзадача: Базовый решатель (3 балла)

Описание: Этот модуль имитирует наиболее очевидные стратегии. Ваш алгоритм должен циклически сканировать все открытые ячейки и применять два простых правила до тех пор, пока можно сделать хотя бы один гарантированный ход.

* Правило 1 (Все мины): Если число в ячейке N равно количеству неоткрытых соседей, все эти соседи — мины.
* Правило 2 (Все безопасны): Если число в ячейке N равно количеству флагов среди соседей, все остальные неоткрытые соседи безопасны.

Требования к реализации:

Вам необходимо реализовать метод solve_step(self, board: list[list[Cell]]) -> tuple[set[tuple[int, int]], set[tuple[int, int]]].

* Метод принимает на вход текущее состояние игрового поля.
* Он должен возвращать кортеж из двух множеств:Множество set координат (row, col), которые гарантированно безопасны для открытия.Множество set координат (row, col), где гарантированно находятся мины (для установки флагов).
* Множество set координат (row, col), которые гарантированно безопасны для открытия.

* Множество set координат (row, col), где гарантированно находятся мины (для установки флагов).
* Метод должен применять Правила 1 и 2 итеративно, пока не перестанет находить новые ходы.

Подзадача: Продвинутый решатель на основе CSP (6 баллов)

Теоретическая справка: Часто базовый решатель заходит в тупик. Такие ситуации эффективно моделируются как Проблема Удовлетворения Ограничений (CSP).

* Переменные: Каждая закрытая ячейка на границе с открытыми является булевой переменной (1 — мина, 0 — безопасно).
* Ограничения: Каждая открытая ячейка с числом N создает линейное ограничение: сумма значений соседних переменных должна равняться N минус количество уже найденных мин.

Алгоритм (Правило подмножеств):

Если набор переменных одного ограничения является подмножеством набора переменных другого, можно вычесть одно уравнение из другого и получить новое, более простое ограничение.

Пример:

* Ограничение 1: x1 + x2 = 1
* Ограничение 2: x1 + x2 + x3 = 2
* Вычитая (1) из (2), получаем: x3 = 1. Это означает, что x3 — мина.

Требования к реализации:

Вам необходимо расширить логику метода solve_step.

1) Реализуйте внутренний метод _generate_constraints, который анализирует поле и формирует список ограничений. Каждое ограничение удобно представить как кортеж: (frozenset[tuple[int, int]], int), где frozenset — неизменяемое множество координат переменных, а int — количество мин среди них.
2) Реализуйте метод _apply_subset_rule, который итерирует по всем парам ограничений и выводит новые факты.

Если в результате применения правила подмножеств были найдены новые безопасные ячейки или мины, добавьте их в соответствующие множества, которые возвращает solve_step.

Часть III - Вероятностный решатель (5 баллов)

Подзадача - обоснованные предположения (5 баллов)

Описание: Когда детерминированный решатель не может сделать ход, необходимо сделать наиболее безопасное предположение. Для этого нужно вычислить точную

вероятность нахождения мины для каждой неоткрытой ячейки.

Требования к именованию: Логика должна быть реализована в методе make_probabilistic_move(self, board: list[list[Cell]]) -> tuple[int, int]. Метод должен возвращать координаты (row, col) ячейки с наименьшей вероятностью мины.

Алгоритм:

1) Идентификация независимых областей: Разделите все пограничные ячейки на независимые группы. Две ячейки связаны, если они входят в одно ограничение. Для этой задачи идеально подходит структура данных Система непересекающихся множеств (Union-Find) или стандартный обход графа (BFS/DFS).
2) Перебор конфигураций: Для каждой независимой области найдите все возможные расстановки мин, которые удовлетворяют всем ограничениям. Это классическая задача, которая решается с помощью рекурсивного алгоритма с возвратом (backtracking).
3) Подсчет вероятностей: Для каждой ячейки Xi в области вычислите вероятность по формулеP(Xi=1) = Общее количество валидных решений / Количество валидных решений, где Xi = 1Этот метод основан на подсчете всех "возможных миров", совместимых с имеющимися данными.
4) Выбор хода: Проанализируйте вероятности для всех пограничных ячеек и верните координаты той, у которой вероятность быть миной минимальна.

Заключение и требования к сдаче

Ожидаемые результаты

По завершении работы вы должны предоставить:

* Работоспособное консольное приложение «Сапёр» (play_minesweeper.py).
* Модуль с логикой игры (minesweeper_engine.py).
* Класс Solver (solver.py), который последовательно применяет все три уровня логики.

Критерии оценки

Описание	Баллы
Структуры данных и состояние: Корректная реализация Cell, генерация поля, безопасный первый ход, предрасчет.	3
CLI и игровая логика: Текстовый интерфейс, обработка команд, итеративное каскадное открытие.	3
Базовый решатель: Корректная реализация solve_step с базовыми правилами.	3
Продвинутый решатель (CSP): Реализация правила подмножеств в	6

solve_step.	
Вероятностный решатель: Реализация make_probabilistic_move с поиском областей, backtracking и расчетом вероятностей.	5

[3] **Реализация системы контроля версий**

Лабораторная работа

Описание:

Введение и теоретическая основа

В этой лабораторной работе вам предстоит погрузиться во внутреннее устройство одной из самых популярных систем контроля версий — Git. Вашей задачей будет не просто использовать Git, а воссоздать его ключевую функциональность с нуля на языке Python.

Вы создадите консольное приложение pygit, которое будет эмулировать основные команды Git: init, add, commit, log и другие. Работа над этим проектом даст вам уникальное понимание того, как Git управляет файлами, историей изменений и ветками на самом низком уровне.

Как устроен Git

Чтобы успешно реализовать pygit, необходимо понимать три фундаментальных концепта, лежащих в основе Git: объектную модель, контентно-адресуемое хранилище и механизм указателей.

1. Объектная модель и контентно-адресуемое хранилище

Вся информация в Git хранится в виде объектов четырех типов. Эти объекты помещаются в специальную директорию .git/objects. Git не хранит разницу между файлами (дельты) в своей основной модели; вместо этого он сохраняет полные снимки (snapshots) состояния проекта.

Ключевая особенность хранилища Git — оно контентно-адресуемое. Это означает, что имя файла, в котором хранится объект, является SHA-1 хешем его содержимого. Например, если содержимое файла — строка "hello world", Git вычислит её SHA-1 хеш, который будет уникальным идентификатором этого контента.

* Blob (Binary Large Object): Самый простой тип объекта. Он хранит точное содержимое файла. Git не интересует имя файла; blob — это просто последовательность байт.
* Tree: Объект-дерево представляет собой аналог директории в файловой системе. Он хранит список указателей на другие blob и tree объекты. Каждая запись в tree

содержит:Права доступа к файлу/директории.Тип объекта (blob или tree).SHA-1 хеш объекта, на который он указывает.Имя файла или поддиректории. Таким образом, tree объекты рекурсивно формируют полный снимок состояния проекта.
* Права доступа к файлу/директории.
* Тип объекта (blob или tree).
* SHA-1 хеш объекта, на который он указывает.
* Имя файла или поддиректории. Таким образом, tree объекты рекурсивно формируют полный снимок состояния проекта.
* Commit: Объект-коммит связывает все воедино. Он представляет собой один снимок в истории проекта. Коммит содержит:SHA-1 хеш корневого tree объекта, который описывает состояние проекта в момент коммита.SHA-1 хеш родительского коммита (или нескольких, в случае слияния). Первый коммит в истории не имеет родителя.Информацию об авторе и коммитере (имя, email, временная метка).Сообщение коммита. Именно коммиты, связанные через своих родителей, образуют историю изменений.
* SHA-1 хеш корневого tree объекта, который описывает состояние проекта в момент коммита.
* SHA-1 хеш родительского коммита (или нескольких, в случае слияния). Первый коммит в истории не имеет родителя.
* Информацию об авторе и коммитере (имя, email, временная метка).
* Сообщение коммита. Именно коммиты, связанные через своих родителей, образуют историю изменений.

2. Индекс (Staging Area)

Индекс — это одна из самых мощных, но часто неправильно понимаемых частей Git. Это временная область, которая находится между вашей рабочей директорией (файлами, которые вы видите и редактируете) и вашим репозиторием (базой данных объектов). Когда вы выполняете команду git add, вы не просто говорите Git "отслеживать этот файл". Вы добавляете текущее состояние файла в индекс, подготавливая его к следующему коммиту. Индекс, по сути, является "конструктором" следующего коммита.

3. Указатели: Ветки и HEAD

Ветки в Git — это не тяжеловесные копии проекта. Ветка — это всего лишь легковесный указатель на определенный коммит. Технически, это просто файл в директории .git/refs/heads/, имя которого — название ветки, а содержимое — SHA-1 хеш коммита, на который она указывает.

HEAD — это еще один указатель, который указывает на текущую активную ветку. Это файл .git/HEAD, который обычно содержит ссылку вида ref: refs/heads/main. Он говорит Git, в какой ветке вы сейчас находитесь и какой коммит является последним в

вашей локальной истории.

Часть I. Ядро репозитория и объектная модель (7 баллов)

На этом этапе вы заложите фундамент вашего pygit, реализовав базовую структуру репозитория и классы для работы с внутренними объектами Git.

Подзадача 1.1: Инициализация репозитория (2 балла)

Требования к именованию: Основной исполняемый файл должен называться pygit_commands.py.

Вам необходимо реализовать команду pygit init. Эта команда должна создавать в текущей директории скрытую папку .pygit, которая будет служить репозиторием. Внутри .pygit должны быть созданы следующие поддиректории и файлы:

* .pygit/objects/: Директория для хранения всех объектов (blob, tree, commit).
* .pygit/refs/heads/: Директория для хранения указателей на ветки.
* .pygit/HEAD: Файл, указывающий на текущую активную ветку. По умолчанию он должен содержать строку ref: refs/heads/main.

Подзадача 1.2: Реализация объектной модели (3 балла)

Требования к именованию: Вся логика для работы с объектами должна находиться в файле pygit/objects.py.

Вам предстоит применить свои знания ООП для создания классов, представляющих внутренние объекты Git.

* Абстрактный базовый класс: Создайте абстрактный базовый класс GitObject с методами serialize() и deserialize().
* Классы объектов: Реализуйте классы Blob, Tree и Commit, наследующиеся от GitObject.Blob(data: bytes): Конструктор принимает содержимое файла в виде байтов. Метод serialize должен возвращать эти байты.Tree: Этот класс должен хранить список записей (например, в виде списка кортежей (mode, path, sha)). Метод serialize должен преобразовывать этот список в специальный бинарный формат, используемый Git.Commit: Должен хранить хеш дерева, хеш родительского коммита, автора и сообщение. Метод serialize должен форматировать эти данные в текстовый вид, аналогичный Git.
* Blob(data: bytes): Конструктор принимает содержимое файла в виде байтов. Метод serialize должен возвращать эти байты.
* Tree: Этот класс должен хранить список записей (например, в виде списка кортежей (mode, path, sha)). Метод serialize должен преобразовывать этот список в специальный бинарный формат, используемый Git.
* Commit: Должен хранить хеш дерева, хеш родительского коммита, автора и сообщение. Метод serialize должен форматировать эти данные в текстовый вид, аналогичный Git.

Подзадача 1.3: Хеширование и сохранение объектов (2 балла)

Требования к именованию: Функциональность должна быть реализована в файле pygit/objects.py.

Реализуйте функцию hash_object(data: bytes, obj_type: str) -> str.

1) Она должна принимать данные объекта и его тип (blob, tree или commit).
2) Формировать заголовок в формате f"{obj_type} {len(data)}\0".
3) Конкатенировать заголовок с данными.
4) Вычислять SHA-1 хеш от полученной строки.
5) (Опционально) Сжимать данные с помощью zlib.
6) Сохранять сжатые данные в .pygit/objects/ по пути, сформированному из хеша (например, хеш de ad beef... сохраняется в .pygit/objects/de/adbeef...).
7) Возвращать вычисленный хеш в виде строки.

Часть II. Индекс и создание коммитов (7 баллов)

На этом этапе вы реализуете основной рабочий процесс: добавление файлов в индекс и создание коммитов.

Подзадача 2.1: Реализация индекса (3 балла)

Требования к именованию: Логика индекса должна быть в pygit/index.py. Команда add реализуется в pygit_commands.py.

Реализуйте команду pygit add <filename>.

1) Прочитайте содержимое файла.
2) Создайте объект blob и сохраните его с помощью hash_object.
3) Реализуйте функции read_index() и write_index(), которые будут работать с файлом .pygit/index. Индекс можно представить как список записей, где каждая запись — это кортеж с информацией о файле (path, sha, mode).
4) Команда add должна обновить или добавить запись о файле в индекс.

Подзадача 2.2: Создание дерева из индекса (2 балла)

Требования к именованию: Реализуйте внутреннюю команду pygit write-tree.

Эта команда должна построить tree объекты на основе текущего состояния индекса.

1) Прочитайте индекс.
2) Сгруппируйте файлы по директориям.
3) Реализуйте генератор, который рекурсивно обходит структуру директорий. Для каждой директории он должен сначала рекурсивно вызывать себя для всех поддиректорий, а затем yield записи для всех файлов в текущей директории.
4) Используя этот генератор, создайте и сохраните все необходимые tree объекты, начиная с самого глубокого уровня вложенности и заканчивая корневым.
5) Команда должна вывести в консоль SHA-1 хеш корневого tree объекта.

Подзадача 2.3: Создание коммитов (2 балла)

Требования к именованию: Реализуйте команду pygit commit -m "<message>".

Эта команда завершает цикл работы, сохраняя снимок проекта.

1) Вызовите логику write-tree, чтобы получить хеш корневого дерева.
2) Определите родительский коммит, прочитав указатель текущей ветки из HEAD.
3) Создайте новый объект Commit с полученным хешем дерева, хешем родителя, информацией об авторе и сообщением из аргумента команды.
4) Сохраните объект Commit с помощью hash_object.

Обновите указатель текущей ветки (например, .pygit/refs/heads/main), чтобы он указывал на хеш нового коммита.

Часть III - История и навигация (6 баллов)

На этом этапе вы добавите возможность просматривать историю и управлять ею, используя итераторы и декораторы.

Подзадача 3.1: Просмотр истории коммитов (3 балла)

Требования к именованию: Реализуйте команду pygit log.

Вам необходимо элегантно реализовать обход истории коммитов.

1) Создайте класс-итератор CommitHistoryIterator.
2) Конструктор итератора должен принимать хеш начального коммита (тот, на который указывает HEAD).
3) Метод __next__ должен:
4) Загружать и десериализовать текущий коммит.
5) Возвращать информацию о нем (хеш, автор, сообщение).
6) Перемещать внутренний указатель на родительский коммит.
7) Если родителей больше нет, вызывать StopIteration.
8) Команда log должна использовать этот итератор в цикле for для вывода истории коммитов от последнего к первому.

Подзадача 3.2: Управление командами (3 балла)

Требования к именованию: Рефакторинг pygit_commands.py.

Ваш pygit_commands.py может стать громоздким из-за разбора аргументов. Давайте сделаем его элегантнее с помощью декораторов.

1) Создайте словарь, который будет хранить соответствие имен команд и функций, их реализующих.
2) Напишите декоратор @command(name), который принимает имя команды в качестве аргумента. Декоратор должен добавлять декорируемую функцию в созданный словарь.
3) Примените этот декоратор ко всем функциям, реализующим команды (init, add, commit, log и т.д.).

Основная логика в pygit_command.py теперь должна просто извлекать имя команды

из sys.argv, находить соответствующую функцию в словаре и вызывать ее, передавая остальные аргументы.

Заключение и требования к сдаче

По завершении работы вы должны предоставить:

* Работоспособное консольное приложение pygit_commands.py, поддерживающее команды init, add, commit, log и write-tree.
* Структурированный проект с модулями pygit/objects.py, pygit/index.py и т.д.
* Код, демонстрирующий уверенное владение ООП, генераторами, итераторами и декораторами.

Критерии оценки

Подзадача	Описание	Баллы
1.1	Инициализация репозитория: Корректное создание структуры .pygit.	2
1.2	Объектная модель: Реализация классов Blob, Tree, Commit с наследованием.	2
1.3	Хеширование и сохранение: Реализация hash_object с контентно-адресуемым хранением.	2
2.1	Индекс: Реализация команд add и функций для работы с файлом индекса.	2
2.2	Создание дерева: Реализация write-tree с использованием генератора.	2
2.3	Создание коммитов: Корректная реализация команды commit.	2
3.1	Просмотр истории: Реализация log с использованием класса-итератора.	2
3.2	Управление командами: Рефакторинг CLI с использованием декоратора.	3
	Кодстайл: соблюдение	3

	требований PEP8 и прочих из документа “Требования к кодстайлу” (доступно по ссылке)	
Итого		20

[4] **Работа с внешним API (Телеграм)**

Лабораторная работа

Описание:

Введение и теоретическая основа

Обзор

Эта лабораторная работа посвящена созданию Telegram-бота с нуля с использованием Telegram Bot API.

Если вы не знакомы с термином API, то рекомендую прочитать статью: What is an API? In English, please

Теоретическая справка

Telegram Bot API — это HTTP-интерфейс, позволяющий разработчикам создавать ботов. Взаимодействие с API происходит путем отправки HTTPS-запросов на специальный URL.

Структура запроса:

https://api.telegram.org/bot<token>/<methodName>

где <token> - уникальный токен авторизации вашего бота, а <methodName> - название метода API, который вы хотите вызвать (например, getMe, sendMessage)

Параметры могут передаваться как в URL (для GET-запросов), так и в теле запроса в формате JSON (для POST-запросов).

Как получить токен для бота

1) Найдите в Telegram бота @BotFather.
2) Отправьте ему команду /newbot и следуйте инструкциям.
3) BotFather пришлет вам токен. Храните этот токен в секрете.

Получение обновлений

Ваш бот должен как-то узнавать о новых сообщениях.

Мы будем использовать метод getUpdates, который работает по принципу "длинных опросов" (long polling). Бот отправляет запрос к серверу Telegram и ждет ответа в

течение заданного времени (например, 30 секунд). Если за это время приходит новое сообщение, сервер немедленно возвращает его. Если нет — возвращает пустой ответ по истечении тайм-аута. Чтобы не получать одни и те же сообщения повторно, используется параметр offset.

Асинхронное программирование (asyncio и aiohttp)

Для создания производительного бота, способного обслуживать множество пользователей одновременно, мы будем использовать asyncio. Вместо синхронной библиотеки requests, которая блокирует выполнение программы на время ожидания ответа от сервера, мы будем использовать aiohttp — библиотеку для совершения асинхронных HTTP-запросов.

Веб-скрапинг (BeautifulSoup)

Для извлечения информации с веб-страниц мы будем использовать библиотеку BeautifulSoup4, которая отлично парсит HTML и позволяет легко находить нужные данные.

Установка необходимых библиотек

```
pip install requests aiohttp beautifulsoup4
```

Основы взаимодействия с API

На этом этапе вы научитесь отправлять запросы к API Telegram и получать обновления, используя синхронный подход.

Настройка и отправка сообщений

Создайте файл sync_bot.py

1) Проверка токена: напишите функцию, которая вызывает метод getMe с помощью библиотеки requests. Она должна отправлять GET-запрос и выводить в консоль информацию о вашем боте, подтверждая, что токен работает.
2) Отправка сообщений: реализуйте функцию send_message(chat_id, text). Она должна отправлять POST-запрос к методу sendMessage. Параметры chat_id и text должны передаваться в теле запроса в формате JSON.

Получение сообщений и эхо-бот

1) Получение обновлений: Реализуйте функцию get_updates(offset=None, timeout=30). Она должна вызывать метод getUpdates и использовать параметры offset и timeout.
2) Создание эхо-бота: Напишите основной цикл работы бота. В бесконечном цикле while True: бот должен:Вызывать get_updates, передавая offset от предыдущего шага.Проходить по списку полученных обновлений.Для каждого сообщения извлекать chat_id и text.Отправлять полученный текст обратно пользователю с

помощью вашей функции send_message.
3) Вызывать get_updates, передавая offset от предыдущего шага.
4) Проходить по списку полученных обновлений.
5) Для каждого сообщения извлекать chat_id и text.
6) Отправлять полученный текст обратно пользователю с помощью вашей функции send_message.

Обновлять offset, чтобы не получать старые сообщения. offset должен быть равен update_id последнего обработанного обновления + 1.

Интеграция с веб-скрапингом

Добавим боту полезную функцию — получение данных со стороннего сайта.

1) Функция-скрапер: Напишите функцию get_daily_quote(). Она должна:С помощью requests загрузить HTML-страницу сайта с цитатами.С помощью BeautifulSoup найти на странице цитату дня и ее автора.Возвращать отформатированную строку с цитатой и автором.
2) С помощью requests загрузить HTML-страницу сайта с цитатами.
3) С помощью BeautifulSoup найти на странице цитату дня и ее автора.
4) Возвращать отформатированную строку с цитатой и автором.
5) Интеграция в бота: В основном цикле вашего эхо-бота добавьте проверку. Если текст сообщения от пользователя — /quote, бот должен вызвать get_daily_quote() и отправить результат пользователю.

Переход к асинхронности

Асинхронный бот

Создайте новый файл async_bot.py

1) Рефакторинг API-функций: Перепишите ваши функции send_message и get_updates на асинхронный лад. Они должны стать async def функциями и использовать aiohttp.ClientSession для выполнения запросов.
2) Асинхронный цикл: Перепишите основной цикл while True: в асинхронную функцию main(). Все вызовы к вашим новым API-функциям должны использовать await. Запустите цикл с помощью asyncio.run(main()).
3) Интеграция скрапера: Адаптируйте команду /quote для работы в асинхронном цикле. Обратите внимание, что requests — синхронная библиотека. Чтобы не блокировать асинхронный цикл, выполните скрапинг в отдельном потоке с помощью asyncio.to_thread (доступно в Python 3.9+).

Конкурентные задачи

Продемонстрируйте мощь асинхронности, выполняя несколько задач одновременно.

1) Несколько скраперов: Напишите 2-3 асинхронные функции-скрапера (например, для получения новостей с разных сайтов). Каждая функция должна использовать aiohttp для загрузки страницы.
2) Команда /headlines: Создайте обработчик для этой команды.
3) Конкурентное выполнение: Внутри обработчика используйте asyncio.gather() для

одновременного запуска всех ваших функций-скраперов.
4) Агрегация результатов: Дождитесь завершения всех задач, соберите результаты, отформатируйте их в одно сообщение и отправьте пользователю.

Управление состоянием

Простой конечный автомат (FSM)

Реализуйте многошаговый диалог без использования сторонних FSM-библиотек.

1) Хранилище состояний: Создайте глобальный словарь user_states = {}, где ключом будет user_id, а значением — текущее состояние пользователя (например, строка 'waiting_for_city').
2) Реализация погоды:Создайте обработчик для команды /weather. Когда пользователь отправляет эту команду, бот должен ответить "Пожалуйста, введите название города." и установить состояние для этого пользователя: user_states[user_id] = 'waiting_for_city'.В основном цикле обработки обновлений добавьте логику: перед обработкой сообщения как обычной команды, проверьте, есть ли пользователь в user_states.Если состояние пользователя — 'waiting_for_city', то текст его сообщения нужно считать названием города.Выполните асинхронный запрос к любому бесплатному API погоды (например, OpenWeatherMap), передав полученный город.Отправьте пользователю отформатированный ответ с погодой.Удалите состояние пользователя из словаря: del user_states[user_id].
3) Создайте обработчик для команды /weather. Когда пользователь отправляет эту команду, бот должен ответить "Пожалуйста, введите название города." и установить состояние для этого пользователя: user_states[user_id] = 'waiting_for_city'.
4) В основном цикле обработки обновлений добавьте логику: перед обработкой сообщения как обычной команды, проверьте, есть ли пользователь в user_states.
5) Если состояние пользователя — 'waiting_for_city', то текст его сообщения нужно считать названием города.
6) Выполните асинхронный запрос к любому бесплатному API погоды (например, OpenWeatherMap), передав полученный город.
7) Отправьте пользователю отформатированный ответ с погодой.
8) Удалите состояние пользователя из словаря: del user_states[user_id].

Критерии оценки

Синхронные запросы: Реализованы getMe и sendMessage через requests.	2
Синхронный эхо-бот: Реализован цикл long polling с управлением offset.	2
Синхронный скрапинг: Реализована команда /quote с использованием requests и BeautifulSoup.	2
Асинхронный бот: Код переписан на asyncio и aiohttp, скрапинг выполняется неблокирующим способом.	3
Конкурентные задачи: Команда	4

/headlines выполняет несколько скрапинг-задач одновременно с помощью asyncio.gather().	
Управление состоянием: Реализован многошаговый диалог для команды /weather с помощью словаря состояний.	4
Кодстайл: соответствие кода требованиям PEP8 и строгой типизации (успешное прохождение кодом линтеров mypy, flake8)	3
Итого	20

Дифференцированный зачет

Регламент проведения дифференцированного зачета

Зачет проводится в устной форме с демонстрацией практических навыков. Билет включает два задания:

1. Теоретический блок (дискуссионный вопрос)

* Цель: Оценка концептуального понимания методологий и архитектурных решений.
* Требования: Студент должен аргументированно изложить преимущества и недостатки подходов, привести релевантные примеры и сделать обоснованные выводы.
* Оценивание (0–10 баллов): Высокий балл выставляется за глубину рассуждений и самостоятельность мышления; снижение балла — за механическое воспроизведение материала без понимания сути.

2. Практический блок (Live Coding)

* Цель: Проверка навыков прикладного программирования и владения инструментарием языка (Python).
* Требования: Написание рабочего кода, решающего поставленную задачу, с пояснением хода решения.
* Оценивание (0–10 баллов): Учитывается работоспособность, логичность, аккуратность кода и способность обосновать выбор реализации.

Порядок сдачи: Студенту предоставляется время на подготовку (5–10 мин). Допускается использование справочных материалов, однако оценка базируется на способности студента вести диалог и решать задачи в реальном времени. Ответ включает устную презентацию (при необходимости — с графическими схемами) и демонстрацию написания кода с комментариями.

Примеры заданий

Билет №1: Вопрос (10 баллов):Обсудите роль протокола итераций и генераторов в повышении эффективности кода при обработке больших объемов данных. Почему ленивые вычисления и отложенное формирование результатов могут быть важнее, чем максимальная скорость выполнения отдельного фрагмента кода? Приведите примеры сценариев, в которых использование итераторов и генераторов оправдано, и порассуждайте, как эти инструменты

позволяют инженеру программного обеспечения находить разумные компромиссы между быстротой, потреблением памяти и удобством сопровождения кода.

Лайвкодинг (10 баллов):Реализуйте простой генератор, который принимает на вход число N и по очереди выдает квадраты чисел от 1 до N, не формируя при этом целый список в памяти. Покажите, как можно использовать этот генератор для пошаговой обработки данных (например, вычислить сумму результатов, не храня все квадраты в памяти).

Билет №2: Вопрос (10 баллов):Обсудите подходы к организации крупных проектов на Python с использованием модулей и пакетов. Почему архитектурные решения, связанные с разбиением на модули и использование пространств имен, важнее простого знания синтаксиса? Предположите ситуацию, когда группа разработчиков работает над большим проектом и вам нужно отстоять определенную модульную архитектуру — какую логику вы будете использовать в споре с коллегами, чтобы убедить их в эффективности вашего подхода?

Лайвкодинг (10 баллов):Напишите небольшой фрагмент кода, который импортирует функционал из другого модуля (предположим, модуль utils.py), содержащего функцию process_data(data_list). В своем основном скрипте используйте эту функцию для обработки списка данных, при этом продемонстрируйте корректную организацию импорта и работу с результатом без использования глобальных переменных. Поясните свои решения по ходу написания кода в комментариях.