МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

УНИВЕРСИТЕТ САТПАЕВ

Институт\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



ЛАБОРАТОРНАЯ РАБОТА

Тема: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| № | Качество выполнения работы | Диапазон  оценки | Получено,  % |
| 1 | Не выполнено | 0% |  |
| 2 | Выполнено | 0-50% |  |
| 3 | Самостоятельная систематизация материала | 0-10% |  |
| 4 | Выполнение требуемого объема и в указанный срок | 0-5% |  |
| 5 | Использование дополнительной научной литературы | 0-5% |  |
| 6 | Уникальность выполненного задания | 0-10% |  |
| 7 | Защита работы | 0-20% |  |
|  | Итого: | 0-100% |  |

Преподаватель Қасенхан Арай

Студент Жаңбырбайқызы Қарлығаш

Группа СР 12:10-14:05

Алматы 2023 г

# Жаңбырбайқызы Қарлығаш

# 4-зертхана: Жетілдірілген рекурсия және функционалдық үлгілер

**Мақсат:**

Python тіліндегі озық рекурсивті әдістер мен функционалдық үлгілерді терең меңгеру және қолдану. Зертхананың мақсаты рекурсивті кодты жазу дағдыларын жақсарту ғана емес, сонымен қатар бағдарламалардың оқылуын, тиімділігін және масштабталуын жақсартатын әртүрлі функционалдық үлгілерді түсіну және қолдану болып табылады.

Тапсырмалар:

1. Рекурсияны терең зерттеу:
   * Күрделі рекурсивті есептер мен алгоритмдерді талдау.
   * Python тілінде құйрықты рекурсия және оны жүзеге асыру принциптерін зерттеу.
   * мәліметтерді өңдеу және алгоритмдік есептерді шешу үшін рекурсияны практикалық қолдану.
2. Функционалдық үлгілерді меңгеру:
   * Монадалар, карриинг және функционалдық композиция сияқты функционалдық үлгілерді зерттеу және қолдану.
   * Функционалдық үлгілердің кодты ұйымдастыруға және құрылымына қалай әсер ететіні туралы түсінікті дамыту.
3. Функционалдық программалауды нақты есептерде қолдану:
   * Практикалық есептерге күрделі функционалды шешімдерді енгізу, мысалы, деректерді талдау, мәтінді өңдеу немесе веб-қызметтерді құру.
   * Модульділік пен кодтың сыналуын жақсарту үшін функционалдық тәсілдерді қолдану.
4. Жақсартылған код өнімділігі және оқылу мүмкіндігі:
   * Рекурсия мен функционалдық заңдылықтардың бағдарламаның жұмысына әсерін талдау.
   * Функционалдық бағдарламалау стилінің артықшылықтары мен кемшіліктерін кодты оқу және техникалық қызмет көрсету тұрғысынан бағалау.
5. Сыни тұрғыдан ойлау және проблемалық талдау дағдыларын дамыту:
   * Күрделі есептерді шешу кезінде сыни талдау мен тәуелсіз ізденістерді ынталандыру.
   * Теориялық білімдерін практикада қолдану және бейімдеу қабілетін дамыту.

Зертханалық жұмыстың маңыздылығы:

Зертханалық жұмыс функционалдық бағдарламалау мен рекурсияда алдыңғы қатарлы дағдыларды дамытуға бағытталған. Бұл студенттерге кодының сапасы мен ауқымдылығын жақсарта отырып, функционалды бағдарламалау принциптерін нақты әлемдегі мәселелерге қалай қолдануға болатынын жақсы түсінуге көмектеседі. Сондай- ақ жұмыс аналитикалық ойлауды дамытуға және әртүрлі бағдарламалық тапсырмалар мен талаптарға бейімделу қабілетіне ықпал етеді.

# Жеке тапсырмалар:

1. Сөмке мәселесін шешуге арналған рекурсиямен динамикалық бағдарламалау
   * Сөмке мәселесін шешу үшін есте сақтау арқылы рекурсияны пайдаланыңыз.
2. def knapsack(weights, values, capacity):  
    memo = {}  
     
    def knapsack\_helper(index, remaining\_capacity):  
    if (index, remaining\_capacity) in memo:  
    return memo[(index, remaining\_capacity)]  
     
    if index == 0 or remaining\_capacity == 0:  
    return 0  
     
    if weights[index - 1] > remaining\_capacity:  
    result = knapsack\_helper(index - 1, remaining\_capacity)  
    else:  
    include = values[index - 1] + knapsack\_helper(index - 1, remaining\_capacity - weights[index - 1])  
    exclude = knapsack\_helper(index - 1, remaining\_capacity)  
    result = max(include, exclude)  
     
    memo[(index, remaining\_capacity)] = result  
    return result  
     
    return knapsack\_helper(len(weights), capacity)  
     
   weights = [2, 3, 4, 5]  
   values = [3, 4, 5, 6]  
   capacity = 5  
   print(knapsack(weights, values, capacity))

# Бағалау критерийлері:

* Жеке есепті шешу үшін код жазу: 1 ұпай
* Қорғау кезінде жазылған кодты түсіндіру және түсіну: 2 ұпай
* Мұғалім таңдаған теориялық сұрақтардың біріне жауап: 1 ұпай

# Дайындық сұрақтары:

1. Жетілдірілген рекурсияда қолданылатын негізгі принциптер мен әдістер қандай?

Продвинутая рекурсия - это подход к решению задач, который расширяет базовые принципы рекурсии, добавляя дополнительные методы или техники для улучшения эффективности, читаемости или обработки определенных сценариев. Основные принципы и методы, используемые в расширенной рекурсии, могут включать следующее:

1. \*\*Мемоизация (запоминание результатов)\*\*: Это техника, при которой результаты выполнения функции для определенных входных данных сохраняются в памяти и затем переиспользуются при повторных вызовах с теми же входными данными. Мемоизация может значительно сократить количество повторных вычислений и улучшить производительность алгоритма.

2. \*\*Динамическое программирование\*\*: Это метод решения задач, который использует принцип оптимизации и разделения задачи на подзадачи, а затем комбинирует результаты подзадач для получения общего решения. В контексте рекурсии это может означать использование мемоизации для сохранения промежуточных результатов и избежания повторных вычислений.

3. \*\*Терминирующие условия\*\*: Это условия, определяющие базовые случаи или случаи завершения рекурсивной функции. В расширенной рекурсии, помимо базовых случаев, могут использоваться дополнительные условия для обработки особых сценариев или предотвращения бесконечной рекурсии.

4. \*\*Обработка ошибок и исключений\*\*: В расширенной рекурсии может потребоваться учет дополнительных сценариев, связанных с обработкой ошибок или исключений, чтобы обеспечить корректную работу алгоритма в различных условиях.

5. \*\*Оптимизация рекурсии\*\*: Это техники или методы, направленные на улучшение производительности рекурсивных алгоритмов, такие как оптимизация хвостовой рекурсии или преобразование рекурсивного вызова в итеративный для снижения накладных расходов на вызов функции.

6. \*\*Разделение задачи на подзадачи\*\*: Это принцип, который заключается в разбиении основной задачи на более мелкие подзадачи, каждая из которых решается независимо. В контексте рекурсии это может означать разбиение задачи на подзадачи, решение каждой из которых происходит путем рекурсивных вызовов.

Эти принципы и методы могут комбинироваться и применяться в зависимости от конкретной задачи и требований к алгоритму для достижения оптимальных результатов.

* + Мақсаты: Студенттердің рекурсия және есте сақтау сияқты кеңейтілген рекурсиялық ұғымдарды түсінуін қамтамасыз ету.

1. Стек толып кетуін болдырмау үшін рекурсивті функцияларды қалай оңтайландыруға болады?

Оптимизация рекурсивных функций для предотвращения переполнения стека - это важный аспект при разработке рекурсивных алгоритмов. Вот несколько методов оптимизации:

1. \*\*Хвостовая рекурсия\*\*: Это особый случай рекурсии, когда рекурсивный вызов является последней операцией в функции. В этом случае компилятор может оптимизировать вызовы, чтобы не сохранять состояние стека между итерациями. Многие языки программирования поддерживают оптимизацию хвостовой рекурсии, что позволяет использовать рекурсию без риска переполнения стека. Если ваш язык поддерживает хвостовую рекурсию, убедитесь, что ваша рекурсивная функция соответствует этому шаблону.

2. \*\*Преобразование в итерацию\*\*: Если хвостовая рекурсия не подходит для вашего случая, вы можете преобразовать рекурсивный алгоритм в итеративный. Итеративные алгоритмы не используют рекурсию и, следовательно, не сталкиваются с проблемой переполнения стека. Для этого требуется переписать алгоритм так, чтобы он использовал циклы вместо рекурсивных вызовов.

3. \*\*Использование структуры данных для хранения промежуточных результатов\*\*: Вместо того чтобы полагаться на стек вызовов для хранения промежуточных результатов, вы можете использовать структуру данных, такую как стек или очередь, для явного хранения этих результатов. Это позволит уменьшить глубину рекурсии и снизить риск переполнения стека.

4. \*\*Увеличение размера стека\*\*: В некоторых языках программирования можно установить максимальный размер стека, который будет использоваться для рекурсивных вызовов. Однако это не всегда рекомендуется, поскольку это может привести к неэффективному использованию ресурсов и повысить вероятность переполнения памяти.

5. \*\*Оптимизация алгоритма\*\*: Иногда можно изменить сам алгоритм таким образом, чтобы он требовал меньше глубины рекурсии. Это может включать в себя изменение порядка обхода данных или оптимизацию базовых случаев для уменьшения количества рекурсивных вызовов.

Выбор конкретного метода оптимизации зависит от специфики задачи, языка программирования и возможностей компилятора или интерпретатора.

* + Мақсаты: Студенттердің рекурсивті функцияларды оңтайландыру, соның ішінде құйрық рекурсиясын және есте сақтауды пайдалану туралы білімдерін тексеру.

1. Функционалдық декоратор үлгісі дегеніміз не және ол Python тілінде қалай қолданылады?

Шаблон функционального декоратора - это концепция в Python, которая позволяет добавлять дополнительное поведение к функции без изменения её основной реализации. Декораторы представляют собой функции высшего порядка, то есть функции, которые принимают другие функции в качестве аргументов и/или возвращают функции.

* + Мақсаты: Студенттің функционалды декоратор үлгісінің түсінігі мен практикалық қолданылуын түсінуін қамтамасыз ету.

1. Python-да жалқау бағалаудың артықшылықтары қандай және олар қалай жүзеге асырылады?

Ленивая оценка (lazy evaluation) - это стратегия вычислений, при которой значения выражений вычисляются только в момент необходимости. В Python ленивая оценка обычно реализуется с помощью генераторов, итераторов и функций-генераторов, а также через использование специальных библиотек, таких как **functools** или **itertools**.

Преимущества ленивой оценки в Python включают:

1. **Эффективное использование ресурсов**: Значения вычисляются только в момент необходимости, что позволяет избежать ненужных вычислений и экономит память и процессорное время.
2. **Обработка потенциально бесконечных последовательностей**: Ленивая оценка позволяет обрабатывать потенциально бесконечные последовательности данных, так как значения вычисляются по мере необходимости.
3. **Улучшение производительности**: Ленивая оценка позволяет работать с данными порциями, что может значительно улучшить производительность и позволить обрабатывать большие объемы данных.
4. **Повышение читаемости кода**: Использование ленивой оценки может сделать код более читаемым и понятным, так как позволяет выражать операции над данными более декларативно и естественно.
   * Мақсаты: Студенттердің жалқау есептеу принциптері және олардың пайдасы туралы түсінігін тексеру.
5. Стратегияның функционалдық үлгісін енгізу және пайдалану жолын түсіндіре аласыз ба?

Функциональная модель стратегии - это паттерн проектирования, который позволяет определять семейство алгоритмов, инкапсулировать каждый из них и делать их взаимозаменяемыми. Этот паттерн состоит из трех основных компонентов: контекста, стратегий и клиента.

1. **Контекст**: Это объект, который содержит ссылку на один из объектов-стратегий и делегирует выполнение работы этой стратегии. Контекст может предоставлять интерфейс для установки и изменения стратегии.
2. **Стратегии**: Это набор классов или функций, представляющих различные алгоритмы. Каждая стратегия реализует общий интерфейс, определенный контекстом, чтобы их можно было взаимозаменять.
3. **Клиент**: Это объект, который использует контекст для выполнения определенной задачи с использованием выбранной стратегии.
   * Мақсаты: Студенттің стратегия үлгісін түсінуін бағалау және кодтың икемділігі мен модульділігін қамтамасыз ету үшін оны қолдану.
4. Рекурсиямен жұмыс істегенде қандай қиындықтар туындауы мүмкін және оларды қалай шешуге болады?

При работе с рекурсией могут возникать следующие проблемы:

1. \*\*Переполнение стека вызовов\*\*: При слишком большой глубине рекурсии может произойти переполнение стека вызовов, что приведет к ошибке "RecursionError" или "StackOverflowError". Эту проблему можно решить, используя хвостовую рекурсию (tail recursion) или преобразовав рекурсивный алгоритм в итеративный.

2. \*\*Повторные вычисления\*\*: В некоторых случаях рекурсивные функции могут многократно вычислять одни и те же значения, что приводит к избыточным вычислениям и уменьшает производительность. Для решения этой проблемы можно использовать мемоизацию (запоминание результатов) или динамическое программирование.

3. \*\*Бесконечная рекурсия\*\*: В случае неправильной реализации или организации условий завершения рекурсивной функции может произойти бесконечный вызов, что приведет к зависанию программы. Для предотвращения бесконечной рекурсии необходимо убедиться в наличии правильных базовых случаев и условий завершения.

4. \*\*Потеря точности\*\*: Некоторые задачи, решаемые с использованием рекурсии, могут столкнуться с потерей точности из-за ограничений на глубину рекурсии или ограничений по памяти. Это может быть особенно актуально при работе с большими или сложными данными. Для решения этой проблемы можно использовать итеративные алгоритмы или алгоритмы, работающие с более компактными структурами данных.

5. \*\*Сложность отладки и понимания\*\*: Рекурсивные функции могут быть сложными для отладки и понимания из-за своей неявной и накладывающейся природы. Для облегчения отладки и понимания рекурсивных функций можно использовать инструменты отладки, такие как трассировка стека вызовов, а также разбивать рекурсивные алгоритмы на более мелкие подзадачи и документировать код четко.

Решение этих проблем требует внимательного проектирования и реализации рекурсивных алгоритмов, а также применения подходящих техник оптимизации и отладки.

* + Мақсаты: Рекурсияға байланысты мүмкін болатын есептерді және оларды шешу жолдарын түсіну.

1. Бағдарламалауда Observer үлгісін қолдануға мысал келтіріңіз.

Шаблон Observer (наблюдатель) является одним из популярных шаблонов проектирования, который используется для установления зависимости между объектами таким образом, что при изменении состояния одного объекта все зависимые от него объекты автоматически оповещаются и обновляются. Этот шаблон часто используется в ситуациях, когда объекты нуждаются в связи с изменениями своего состояния.

Пример использования шаблона Observer может быть в реализации GUI-фреймворков, где виджеты (наблюдатели) должны быть оповещены о изменениях в модели (издателе), чтобы обновить свое состояние и отобразить новые данные.

* + Мақсаты: Оқушылардың бақылаушы үлгісінің принциптері мен практикалық қолданылуын түсінуін қамтамасыз ету.

1. Рекурсия көмегімен қандай есептер тиімді шешіледі?

Рекурсия - мощный инструмент в программировании, и с ее помощью можно эффективно решать множество задач. Вот некоторые из них:

1. \*\*Обход деревьев и графов\*\*: Рекурсия идеально подходит для обхода деревьев и графов в глубину (DFS) или в ширину (BFS). Например, поиск в глубину, поиск в ширину, поиск кратчайшего пути, поиск путей с заданными условиями и т. д.

2. \*\*Различные комбинаторные задачи\*\*: Например, задачи о перестановках, сочетаниях, разбиениях, распределениях и т. д.

3. \*\*Поиск вариантов\*\*: Рекурсия хорошо подходит для задач, связанных с перебором всех возможных вариантов, например, в задачах о комбинаторике или задачах на дискретную оптимизацию.

4. \*\*Работа с рекурсивными структурами данных\*\*: Рекурсивные структуры данных, такие как списки, деревья, графы, могут быть эффективно обработаны с использованием рекурсии. Например, реализация операций вставки, удаления, поиска в дереве или графе.

5. \*\*Алгоритмы сортировки и поиска\*\*: Некоторые алгоритмы сортировки и поиска также могут быть реализованы с использованием рекурсии, например, быстрая сортировка, сортировка слиянием, бинарный поиск и т. д.

6. \*\*Обработка строк и текста\*\*: В некоторых случаях рекурсия может быть полезна для обработки строк и текста, например, при реализации алгоритмов обработки языков или решения задач обработки текста.

7. \*\*Решение задач динамического программирования\*\*: Многие задачи динамического программирования могут быть решены с использованием рекурсивного подхода, например, задача о рюкзаке, задача о наибольшей общей подпоследовательности и т. д.

Хотя рекурсия может быть мощным инструментом, важно помнить о возможных проблемах, таких как переполнение стека вызовов или избыточные вычисления, и применять ее с умом, особенно при работе с большими или сложными данными.

* + Мақсаты: Оқушылардың рекурсия оңтайлы шешім болатын сценарийлерді анықтау қабілетін тексеру.

1. Рекурсия программаның өнімділігіне және жадты пайдалануға қалай әсер етеді?

Рекурсия может оказывать влияние на производительность программы и использование памяти в зависимости от конкретной реализации и характера задачи. Вот некоторые аспекты, которые следует учитывать:

1. \*\*Производительность\*\*:

- Рекурсивные алгоритмы могут быть менее эффективными, чем итеративные алгоритмы, из-за накладных расходов на вызов функций и управление стеком вызовов.

- Некоторые рекурсивные алгоритмы могут иметь экспоненциальную сложность времени выполнения, что может привести к значительному увеличению времени выполнения программы, особенно при обработке больших или сложных данных.

- Однако в некоторых случаях рекурсия может быть более естественной и понятной в реализации, что важно для поддержания читаемости и поддерживаемости кода.

2. \*\*Использование памяти\*\*:

- Каждый вызов функции в рекурсивном алгоритме создает новый кадр стека вызовов, что требует дополнительной памяти.

- В случае глубокой рекурсии или рекурсивных вызовов с большими объемами данных может произойти переполнение стека вызовов, что может привести к ошибке "RecursionError" или "StackOverflowError".

- Однако некоторые языки программирования, такие как Python, имеют механизм оптимизации хвостовой рекурсии (tail recursion), который позволяет выполнить рекурсивные вызовы без создания новых кадров стека, что снижает использование памяти.

В целом, при использовании рекурсии важно оценивать ее влияние на производительность и использование памяти, а также применять ее там, где это оправдано и уместно. В некоторых случаях рекурсия может быть элегантным и эффективным решением, но в других случаях лучше использовать итеративные или другие алгоритмы для обеспечения оптимальной производительности и использования ресурсов.

* + Мақсаты: Оқушылар рекурсияның жүйе ресурстары мен өнімділігіне әсерін түсінеді.

1. Функционалдық үлгілер бағдарламалық жасақтама дизайны мен архитектурасын қалай жақсарта алатынын түсіндіріңіз.

Функциональные шаблоны - это наборы bewbrf кода и структур, которые часто встречаются в разработке программного обеспечения и которые предназначены для решения определенных типов задач или для достижения определенных целей. Вот несколько способов, как функциональные шаблоны могут улучшить дизайн и архитектуру программного обеспечения:

1. \*\*Увеличение повторного использования кода\*\*: Функциональные шаблоны представляют собой протестированные и оптимизированные способы решения типовых проблем или задач. Используя эти шаблоны, разработчики могут повторно использовать проверенные и надежные методы решения проблем, что помогает уменьшить дублирование кода и снизить вероятность ошибок.

2. \*\*Снижение сложности программы\*\*: Хорошие функциональные шаблоны обеспечивают абстракцию и сокрытие деталей реализации, что позволяет сосредоточиться на более высоком уровне абстракции и упростить понимание и модификацию программы. Это помогает снизить сложность программы и улучшить ее поддерживаемость.

3. \*\*Соблюдение принципов проектирования\*\*: Функциональные шаблоны часто соответствуют принципам проектирования, таким как принцип единственной ответственности, открытости/закрытости, подстановки Барбары Лисков и т. д. Использование этих шаблонов помогает обеспечить соблюдение этих принципов и создать гибкую и расширяемую архитектуру.

4. \*\*Повышение производительности и эффективности\*\*: Функциональные шаблоны могут быть оптимизированы для повышения производительности и эффективности программы. Это может включать в себя использование оптимальных алгоритмов, структур данных и техник оптимизации, что помогает улучшить производительность программы.

5. \*\*Стандартизация подходов к решению задач\*\*: Использование функциональных шаблонов может способствовать стандартизации подходов к решению типовых задач и проблем, что улучшает согласованность кода и уменьшает различия в стиле программирования между разработчиками.

6. \*\*Повышение уровня абстракции\*\*: Функциональные шаблоны позволяют абстрагироваться от конкретных деталей реализации и думать на более высоком уровне абстракции. Это способствует созданию более гибких и адаптивных систем, которые легче поддаются изменениям и расширениям.

В целом, функциональные шаблоны являются мощным инструментом в проектировании программного обеспечения, который помогает улучшить его качество, гибкость, поддерживаемость и эффективность.

# Пример для решения

Задача: Реализация Функции Каррирования для Математических Операций

Каррирование - это процесс преобразования функции, которая принимает несколько аргументов в функцию, которая принимает первый аргумент и возвращает функцию, принимающую следующий аргумент, и так далее. Целью этой задачи является реализация функции каррирования для простых математических операций, таких как сложение, умножение и вычитание.

Цель задачи:

Написать функцию каррирования, которая преобразует двухаргументную математическую функцию в последовательность функций с одним аргументом.

Решение:

def curry(binary\_function): """

Функция каррирования для двухаргументной функции. """

def f(first\_arg):

def g(second\_arg):

return binary\_function(first\_arg, second\_arg) return g

return f

# Пример использования def add(x, y):

return x + y

def multiply(x, y): return x \* y

# Каррированные версии функций curried\_add = curry(add) curried\_multiply = curry(multiply)

# Использование каррированных функций add\_five = curried\_add(5)

print(add\_five(3)) # Результат: 8 (5 + 3)

multiply\_by\_two = curried\_multiply(2) print(multiply\_by\_two(4)) # Результат: 8 (2 \* 4)

Объяснение:

Функция `curry` принимает двухаргументную функцию `binary\_function` и возвращает новую функцию `f`. Функция `f` принимает первый аргумент и возвращает ещё одну функцию `g`, которая принимает второй аргумент и выполняет исходную двухаргументную функцию `binary\_function` с этими двумя аргументами.

Этот подход позволяет преобразовать функции, требующие несколько аргументов, в цепочку функций с одним аргументом, что является классическим примером использования каррирования в функциональном программировании. Это упрощает создание специализированных функций на основе общих и повышает гибкость при составлении сложных выражений.