**3-зертхана: Классикалық есептер үшін рекурсивті функцияларды орындау**

**Мақсат:**

Классикалық алгоритмдік және математикалық есептерді шешу үшін Python тілінде рекурсивті функцияларды үйреніңіз және қолданыңыз. Жұмыстың мақсаты студенттердің рекурсивті кодты жазу, оның принциптері мен шектеулерін түсіну, сонымен қатар рекурсивті алгоритмдерді талдау және оңтайландыру дағдыларын дамыту болып табылады.

Тапсырмалар:

1. Рекурсия негіздері:

* Рекурсия ұғымын және оның итеративті тәсілден айырмашылығын түсіну.
* Рекурсивті функциялардың негізгі мысалдарын оқу.

2. Рекурсивті функциялардың дамуы:

* Классикалық рекурсивті есептерді орындау, мысалы, факториалды, Фибоначчи сандарын, екілік іздеуді және т.б.
* Итерациялық әдістермен салыстырғанда рекурсивті функциялардың тиімділігін талдау.

3. Қоңыраулар стегі мен рекурсиялық шектеулерді түсіну:

* Python тіліндегі шақыру стек механизмін және оның рекурсивті функцияларға әсерін зерттеу.
* Қоңыраулар стекінің толып кету мәселесін анықтау және болдырмау.

4. Рекурсивті функцияларды оңтайландыру:

* Рекурсивті қоңырауларды оңтайландыру үшін есте сақтау және динамикалық бағдарламалау әдістерін пайдалану.
* оңтайландырылған рекурсивті функциялардың өнімділігін олардың оңтайландырылмаған нұсқаларымен салыстыру.

5. Практикалық есептердегі рекурсияның қолданылуы:

- Ағаштарды, графиктерді және басқа деректер құрылымдарын аралау сияқты рекурсивті тәсілді қажет ететін күрделірек мәселелердің шешімдерін әзірлеу.

Зертханалық жұмыстың маңыздылығы:

Бұл зертхана студенттерге рекурсияны бағдарламалаудағы қуатты құрал ретінде түсінуді тереңдетуге көмектеседі. Итерациялық әдістерді қолдану арқылы шешуі қиын немесе тиімсіз есептерді шешу үшін жиі рекурсивті әдістер қолданылады. Рекурсияны меңгеру программисттің құралдар жиынтығын кеңейтіп, алгоритмдік есептердің кең ауқымын тиімді шешуге мүмкіндік береді.

**Жеке тапсырмалар:**

Әрбір студентке топ тізіміндегі санына сәйкес бірегей тапсырма беріледі (SSO қараңыз). Студенттерге рекурсияға, деректер құрылымдарын айналып өтуге және рекурсивті функцияларды оңтайландыруға көмектесетін есептер

1. Факториалды есептеу

- Санның факториалын есептеу үшін рекурсивті функцияны орындау.

2. Фибоначчи сандары

- \(n\)-ші Фибоначчи санын есептеу үшін рекурсивті функцияны жазыңыз.

3. Тізім элементтерінің қосындысы

- Тізімдегі элементтердің қосындысын табу үшін рекурсивті функцияны орындаңыз.

4. Ағаштарды аралау

- Екілік ағашты рекурсивті айналдыру және оның элементтерін шығару функциясын жазыңыз.

5. Сызықты кері айналдырыңыз

- Жолды кері қайтару үшін рекурсивті функцияны орындаңыз.

6. Ең үлкен ортақ бөлгішті табу (GCD)

- Екі санның gcd мәнін табу үшін рекурсивті функцияны жазыңыз.

7. Ханой мұнаралары

- Рекурсияның көмегімен Ханой мұнаралары мәселесін шешіңіз.

8. Палиндромды тексеру

- Жолдың палиндром екенін тексеру үшін рекурсивті функцияны орындаңыз.

9. Санның дәрежесін есептеу

- Санның дәрежесін есептейтін рекурсивті функцияны жаз.

1. Тізімдегі элементті табу
   * Сұрыпталмаған тізімдегі элементті рекурсивті іздеуді жүзеге асыру.
2. Комбинацияларды есептеу
   * \( n \) бастап \( k \) аралығындағы комбинациялар санын рекурсивті есептеу функциясын жазыңыз.
3. Рекурсивті сұрыптау
   * Біріктірілген сұрыптау сияқты рекурсивті сұрыптау алгоритмін жүзеге асыру.
4. Графикті өту
   * Графикті рекурсивті айналдыру функциясын жазыңыз (мысалы, тереңдікбірінші іздеу).

def dfs(g, a):  
 t = set()  
 def f2(z):  
 t.add(z)  
 print(z)  
 for b in g[z]:  
 if b not in t:  
 f2(b)  
 f2(a)  
g = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': [],  
 'E': ['F'],  
 'F': []  
}  
dfs(g, 'A')

1. Лабиринттегі рекурсивті жолды табу
   * Рекурсияның көмегімен лабиринттегі жолды табу функциясын орындаңыз.
2. Факториалды есте сақтау
   * Есте сақтау арқылы рекурсивті факторлық есептеу функциясын оңтайландыру.

**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. Рекурсия дегеніміз не және ол бағдарламалау контекстінде қалай жұмыс істейді?

- Мақсаты: Оқушылардың рекурсия ұғымы туралы жалпы түсінігін тексеру.

2. Рекурсивті функцияның негізгі компоненттері қандай?

- Мақсаты: Студенттің рекурсивті функцияның құрылымын, оның ішінде негізгі жағдай мен рекурсивті қадамды түсінуін қамтамасыз ету.

3. Бағдарламалауда рекурсияны қолданудың артықшылықтары мен кемшіліктері қандай?

- Мақсаты: Оқушылардың рекурсияның артықшылықтарын (мысалы, кодтың қарапайымдылығы) және кемшіліктерін (мысалы, стектің толып кету қаупі) түсінуін бағалау.

4. Рекурсия ең жақсы шешім болып табылатын мәселеге мысал келтіріңіз.

- Мақсаты: Студент рекурсия тиімді тәсіл болатын сценарийлерді анықтай алатынын көру.

5. Рекурсияда қоңыраулар стекінің толып кетуін қалай болдырмауға болады?

- Мақсаты: Студент есте сақтау немесе итеративті тәсілге көшу сияқты терең рекурсиямен байланысты мәселелерді болдырмау әдістерін түсінеді.

6. Құйрық рекурсиясы дегеніміз не және ол не үшін маңызды?

- Мақсаты: Студенттердің құйрық рекурсиясы түсінігі және оның артықшылықтары туралы білімдерін тексеру.

7. Рекурсивті функцияда есте сақтау қалай жүзеге асырылады?

- Мақсаты: есте сақтау сияқты рекурсивті қоңырауларды оңтайландыру механизмдерін түсінуді бағалау.

8. Есептерді шешудің рекурсивті және итеративті тәсілдерін салыстыра аласыз ба?

- Мақсаты: Рекурсивті және итеративті әдістердің айырмашылығын түсіну және оларды салыстыра білу.

9. Рекурсивті функцияларды жазу кезінде қандай қателер болуы мүмкін және оларды қалай болдырмауға болады?

- Мақсаты: Студенттің негізгі регистрді жіберіп алу немесе дұрыс емес күй өзгерістері сияқты жиі кездесетін қателерді білуін қамтамасыз ету.

10. Ағаштар мен графиктер сияқты деректер құрылымдарын айналып өту үшін рекурсияны қалай пайдалануға болады?

- Мақсаты: Студенттердің күрделі деректер құрылымдарын өту үшін рекурсияны пайдалану туралы түсінігін тексеру.

**Примерная Задача для Лабораторной Работы 3**

**Задача: Рекурсивное Решение Задачи о Ханойских Башнях**

Задача о Ханойских башнях — классическая задача, решаемая с использованием рекурсии. Цель состоит в том, чтобы переместить диски с одного стержня на другой, придерживаясь определенных правил.

Правила:

1. За один раз можно перемещать только один диск. 2. Диск можно класть только на пустой стержень или на диск большего размера.

Цель задачи:

Написать рекурсивную функцию, которая выводит последовательность шагов для решения задачи о Ханойских башнях для n дисков.

Решение:

def hanoi\_towers(n, source, target, auxiliary):

"""

Решение задачи о Ханойских башнях.

n - количество дисков source - начальный стержень target - целевой стержень

auxiliary - вспомогательный стержень

""" if n > 0:

# Переместить n-1 дисков на вспомогательный стержень hanoi\_towers(n-1, source, auxiliary, target)

# Переместить оставшийся диск на целевой стержень print(f"Переместить диск с {source} на {target}")

# Переместить n-1 дисков с вспомогательного на целевой стержень hanoi\_towers(n-1, auxiliary, target, source)

# Пример использования функции

hanoi\_towers(3, 'A', 'C', 'B')

Объяснение:

Функция `hanoi\_towers` решает задачу о Ханойских башнях, используя рекурсивный подход.

* Если `n` больше 0, сначала перемещаем `n-1` дисков на вспомогательный стержень.
* Затем перемещаем самый большой диск на целевой стержень.
* После этого перемещаем `n-1` дисков с вспомогательного стержня на целевой, следуя тем же правилам.

Эта задача иллюстрирует классическое применение рекурсии и помогает студентам лучше понять концептуальные основы и практическую реализацию рекурсивных функций.

**Дайындық сұрақтары:**

1. Жетілдірілген рекурсияда қолданылатын негізгі принциптер мен әдістер қандай?

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

1. Жетілдірілген рекурсияда қолданылатын негізгі принциптер мен әдістер:

- Хвостовая рекурсия: Рекурсиялық шақырманы функцияның соңында орындау қалпына айналдыратын техника. Кейбір компиляторлардың жасаушылары оны циклге айналдырады.

- Мемоизация: Болдырмауланған жасауларды белгілеу үшін оларды жүзеге асыру жолы. Сыртқы рекурсия мен итерацияны оңтайлау үшін пайдаланылады.

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

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

2. Стек толып кетуін болдырмау үшін рекурсивті функцияларды қалай оңтайландыру:

- Хвостовая рекурсия: Рекурсивті функцияны итерациялық қалпына айналдыру арқылы стек переполнениесін болдырмау.

- Мемоизация: Жауаптарды дайындайтын жасау мен бұл жасауды келесі жасауда пайдалану арқылы дайындату.

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

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

3. Функционалдық декоратор үлгісі:

- Таным: Декоратор - бұл функция, басқа функцияны алатып арттыру үшін оны қолданушының кодын өзгерту мүмкіндігімен.

- Python-де қолдану: @decorator синтаксисі арқылы пайдаланушы кодын дайындау, кэширу, уақытты измеру т.б. мақсаттары үшін пайдаланылады.

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

- Мақсаты: Студенттердің жалқау есептеу принциптері және олардың пайдасы туралы түсінігін тексеру.

4. Python-да жалқау бағалау артықшылықтары:

- Ресурстарды азайту: Жалқау жасау принциптерінен және олардың қолдауынан ресурстарды азайту.

- Өсімдіктер: Python-да функционалдық өсімдіктер, мысалы, map, filter, reduce жаттыру операторларын пайдалану.

5. Стратегияның функционалдық үлгісін енгізу және пайдалану жолын түсіндіре аласыз ба?

- Мақсаты: Студенттің стратегия үлгісін түсінуін бағалау және кодтың икемділігі мен модульділігін қамтамасыз ету үшін оны қолдану.

5. Стратегияның функционалдық үлгісін енгізу және пайдалану:

- Таным: Стратегия - бұл алгоритмдер жиынтығын анықтау, көмек ету және оларды ауыстыру болады.

- Пайдалану: Клиент кодына берілетін мүмкіндіктерді іске асыруда программаны орнына келтіретін стратегияны таңдау.

6. Рекурсиямен жұмыс істегенде қандай қиындықтар туындауы мүмкін және оларды қалай шешуге болады?

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

6. Рекурсиямен жұмыс істегенде қандай қиындықтар туындауы мүмкін және оларды қалай шешуге болады:

- Стек переполненсі: Стекті толып кету тәжірибесінен туындау үшін хвостовая рекурсияны қолдану немесе итерациялық методтарды пайдалану.

- Кайталамалы есептер: Кейбір рекурсивті алгоритмдер кайталамалы жаттыруларды болуы мүмкін. Мемоизацияны қолдану арқылы шешім.

7. Бағдарламалауда Observer үлгісін қолдануға мысал келтіріңіз.

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

7. Бағдарламалауда Observer үлгісін қолдану:

- Мысал: Событтық жүйедегі объекттерді (бақылаушыларды) объектте (табыстырылатын) оқиуге жауап беру және олардың мәліметтерін жаңарту.

8. Рекурсия көмегімен қандай есептер тиімді шешіледі?

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

8. Рекурсия көмегімен қандай есептер тиімді шешіледі:

- Графтар мен деректерді өткізу: Бұларды обход қылу үшін DFS немесе BFS пайдалану.

- Факториалды қарау: n! = n \* (n-1)!.

- \*Тізімдерді сұрау жасау:\* Байланыс бар тізімді сұрау жасау үшін рекурсивті алгоритмдер.

9. Рекурсия программаның өнімділігіне және жадтыр пайдалануға қалай әсер етеді?

- Мақсаты: Оқушылар рекурсияның жүйе ресурстары мен өнімділігіне әсерін түсінеді.

9. \*\*Рекурсия программаның өнімділігіне және жадты пайдалануға қалай әсер етеді:\*\*

- \*Жүйе ресурстарына әсері:\* Рекурсивті функциялар стек памкасын өзгертеді, буында барлық вызовдар стекті толып алады.

- \*Өнімділігіне әсері:\* Мемоизацияны пайдалану арқылы есептердін өтіп, бұл жауаптарды сақтау арқылы ресурстарды өтіп отырарып, өнімділікпен ескеру.

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

10. \*\*Функционалдық үлгілер бағдарламалық жасақтама дизайны мен архитектурасын қалай жақсарта алатынын түсіндіріңіз:\*\*

- \*Дизайны жасау:\* Функционалды үлгілер бағдарламаны кез келген жерде жаңа функционалды қосу жолымен жасауға мүмкіндік береді.

- \*Байланысты іске асыру:\* Үлгілерді байланысты жасағанда жасаушы кодында көздерді байланыс жасауға болатын мүмкіндік береді.

- \*Тұрғынырлы дайындалу:\* Функционалды үлгілер, аймақты үздік жаттығу жасау үшін пайдаланылады.

- \*Жасаушылық және өзара айырыстырылу:\* Функционалды үлгілер арқылы жасаушылық және мәнділікті дайындарды көрсету мүмкін.

1. Продвинутая рекурсия:

- Хвостовая рекурсия: Это концепция, при которой рекурсивный вызов является последней операцией в функции. Использование хвостовой рекурсии позволяет оптимизировать рекурсивные вызовы в итерации.

- Мемоизация: Техника, при которой результаты выполненных вычислений сохраняются для предотвращения повторных вычислений. Это особенно полезно в случае рекурсивных функций с большим количеством повторных вычислений.

2. Оптимизация рекурсивных функций:

- Хвостовая рекурсия: Преобразование рекурсивной функции так, чтобы рекурсивный вызов был последней операцией перед возвратом. Это позволяет некоторым компиляторам преобразовывать рекурсию в цикл.

- Мемоизация: Сохранение результатов выполненных вычислений в структуре данных, такой как словарь, для избежания повторных вычислений.

3. Функциональный паттерн декоратора:

- Определение: Декоратор - это функция, которая принимает другую функцию и добавляет или изменяет её функциональность.

- Применение в Python: Декораторы используются для изменения поведения функций. Например, они могут добавлять логирование, кэширование или изменять возвращаемое значение.

4. Преимущества ленивых вычислений в Python:

- Экономия ресурсов: Вычисления выполняются только по мере необходимости, что снижает использование ресурсов.

- Обработка бесконечных последовательностей: Ленивые вычисления позволяют эффективно работать с бесконечными или большими данными.

5. Реализация функционального паттерна "Стратегия":

- Определение: Стратегия - это семейство алгоритмов, инкапсулированных таким образом, что они становятся взаимозаменяемыми.

- Применение: Реализация паттерна "Стратегия" позволяет выбирать алгоритм во время выполнения программы, что обеспечивает гибкость и модульность кода.

6. Сложности при работе с рекурсией и их решения:

- Переполнение стека: Проблема, возникающая при глубокой рекурсии. Решение: использование хвостовой рекурсии или итеративных методов.

- Повторные вычисления: Некоторые рекурсивные функции могут повторно вычислять те же значения. Решение: мемоизация для сохранения результатов.

7. Пример использования паттерна "Наблюдатель":

- Пример: Реализация системы событий, где объекты (наблюдатели) следят за изменениями в объекте (издателе) и реагируют соответствующим образом.

8. Задачи, наиболее эффективно решаемые с помощью рекурсии:

- Обход деревьев и графов: Глубокий и широкий обход, поиск в глубину (DFS), поиск в ширину (BFS).

- Разделение задачи: Сортировка (например, сортировка слиянием), поиск (например, бинарный поиск).

9. Влияние рекурсии на производительность и использование памяти:

- Производительность: Глубокая рекурсия может привести к переполнению стека. Оптимизации, такие как хвостовая рекурсия и мемоизация, могут помочь снизить риск переполнения.

- Использование памяти: Каждый вызов функции добавляет новый фрейм в стек вызовов, что может потреблять много памяти. Мемоизация помогает сократить повторное выделение памяти.

10. Улучшение проектирования с использованием функциональных паттернов:

- Чистота кода: Функциональные паттерны, такие как декораторы, могут сделать код более чистым, разделяя различные аспекты функциональности.

- Модульность и гибкость: Паттерны, такие как "Стратегия", позволяют разделить сложные алгоритмы на отдельные стратегии, что улучшает модульность и гибкость системы.

- Повторное использование: Функциональные паттерны способствуют повторному использованию кода, поскольку они изолируют функциональные аспекты и алгоритмы.