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

# 5-зертхана: Жалқау есептеулер және функционалдық деректер құрылымдары

**Мақсат:**

Python тіліндегі жалқау бағалау және функционалды деректер құрылымдары ұғымдарын үйреніңіз және қолданыңыз. Зертханалық жұмыстың мақсаты - жалқау кодты орындау принциптерін түсіну, нақты функционалды деректер құрылымдарымен танысу және оларды нақты бағдарламалау тапсырмаларында тиімді пайдалану дағдыларын дамыту.

Тапсырмалар:

1. Жалқау есептеулерді түсіну:
   * Жалқау бағалау түсінігін және оның қатаң (жігерлі) бағалаудан айырмашылығын зерттеу.
   * Генераторлар және генератор өрнектері сияқты Python тіліндегі жалқау бағалау мысалдарын қарастырыңыз.
2. Практикалық есептердегі жалқау есептеулерді қолдану:
   * Жалқау бағалау тиімділікті арттыратын мәселелердің шешімдерін әзірлеу, мысалы, үлкен көлемдегі деректерді өңдеу тапсырмаларында.
   * Жалқау бағалаудың өнімділікке және жадты тұтынуға әсерін зерттеу.
3. Функционалдық деректер құрылымдарын меңгеру:
   * өзгермейтін тізімдер, ағаштар және карталар сияқты дәстүрлі деректер құрылымдарының функционалдық аналогтарын зерттеу.
   * Python тілінде өзгермейтін деректер құрылымдарын пайдалану және құру мүмкіндігін дамыту.
4. Жалқау бағалау және функционалдық құрылымдары бар мысалдарды іске асыру:
   * Жалқау бағалауды және деректердің функционалдық құрылымдарын пайдалануды көрсететін мысалдар мен мәселелердің шешімдерін жасаңыз.
   * Әртүрлі бағдарламалау контекстіндегі мұндай әдістердің артықшылықтары мен шектеулерін талдау.
5. Аналитикалық ойлау мен есептерді шешу дағдыларын дамыту:
   * Бағдарламалаудың жаңа тәсілдерімен тәуелсіз ізденістер мен эксперименттерді ынталандыру.
   * Сыни тұрғыдан бағалау және нақты тапсырмалар үшін сәйкес құралдар мен әдістерді таңдау қабілетін дамыту.

Зертханалық жұмыстың маңыздылығы:

Бұл зертхана студенттердің функционалдық бағдарламалау мүмкіндіктері мен артықшылықтары, соның ішінде жалқау бағалау және функционалды деректер құрылымдары туралы түсініктерін кеңейтуге арналған. Ол деректердің үлкен көлемімен тиімді жұмыс істеу дағдыларын дамытуға және таза, модульдік кодты жасауға бағытталған. Сонымен қатар, жұмыс бағдарламалық есептерді шешу кезінде аналитикалық дағдылар мен ойлау икемділігін арттыруға көмектеседі.

# Жеке тапсырмалар:

Әрбір студентке топ тізіміндегі санына сәйкес бірегей тапсырма беріледі (SSO қараңыз).

1. Жай сандарды жалқау есептеу
   * Жай сандарды жалқаулықпен жасайтын генератор жасаңыз.

def prime\_generator():  
 num = 2  
 while True:  
 is\_prime = True  
 for i in range(2, int(num \*\* 0.5) + 1):  
 if num % i == 0:  
 is\_prime = False  
 break  
 if is\_prime:  
 yield num  
 num += 1  
  
num\_primes = int(input("Жай сан: "))  
  
prime\_gen = prime\_generator()  
for \_ in range(num\_primes):  
 print(next(prime\_gen))

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

# 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. Python-да жалқау бағалау деген нені білдіреді және ол қалай жүзеге асырылады?
   * Мақсаты: Студенттің жалқау бағалау түсінігін және оны Python тілінде қалай енгізу керектігін түсінуін қамтамасыз ету.

Ленивая оценка (lazy evaluation) в Python - это стратегия вычислений, при которой значения выражений вычисляются только тогда, когда они действительно нужны. В контексте языка программирования Python ленивая оценка обычно реализуется с использованием генераторов, отложенных вычислений и специальных структур данных, таких как списки, которые могут быть вычислены по запросу.

Вот как это работает на практике:

1. **Генераторы**: Генераторы позволяют создавать последовательности значений "по запросу", то есть значения генерируются при каждом обращении к ним. Вместо того, чтобы генерировать все значения сразу и хранить их в памяти, генераторы создаются таким образом, чтобы возвращать значения по мере необходимости. Например, **range()** в Python 3 является ленивым генератором, который создает последовательность чисел по запросу.
2. **Отложенные вычисления**: В Python можно использовать отложенные вычисления, например, с помощью лямбда-функций или функций **functools.partial()**. Это позволяет передавать функции аргументы, но не вычислять их значения до тех пор, пока они не понадобятся.
3. **Специальные структуры данных**: В Python есть специальные структуры данных, такие как списки, которые могут быть вычислены по запросу. Например, генераторные выражения и списковые выражения могут быть использованы для создания списков "по запросу", когда значения требуются, а не заранее.
4. Жалқау бағалаудың қандай артықшылықтары бар?

Ленивая оценка имеет несколько преимуществ, особенно в контексте языков программирования, поддерживающих эту стратегию вычислений, таких как Python. Ниже приведены основные преимущества ленивой оценки:

1. **Экономия ресурсов**: Ленивая оценка позволяет избегать вычислений и создания значений, которые не используются в программе. Это позволяет экономить память и процессорное время, особенно при работе с большими или бесконечными структурами данных.
2. **Улучшение производительности**: За счет того, что значения вычисляются только по мере необходимости, можно улучшить производительность программы, особенно если некоторые вычисления являются ресурсоемкими.
3. **Работа с бесконечными структурами данных**: Ленивая оценка позволяет эффективно работать с бесконечными или потенциально бесконечными структурами данных. Например, генераторы в Python позволяют работать с потенциально бесконечными последовательностями без необходимости генерировать все их элементы заранее.
4. **Повышение модульности и улучшение абстракции**: Ленивая оценка позволяет разделять код на более мелкие функции или генераторы, которые могут вычислять значения только по мере необходимости. Это улучшает модульность кода и позволяет создавать более абстрактные и гибкие решения.
5. **Работа с потоками данных**: Ленивая оценка полезна для работы с потоками данных или обработки больших объемов данных. Она позволяет эффективно обрабатывать данные по мере их поступления, без необходимости загрузки всех данных в память.
6. **Интуитивная логика программирования**: Ленивая оценка соответствует интуитивной логике программирования в некоторых случаях. Например, когда необходимо обработать только часть данных или выполнить вычисления только при запросе пользователя.

Эти преимущества делают ленивую оценку мощным инструментом в языках программирования, поддерживающих эту стратегию вычислений, и позволяют эффективно работать с разнообразными задачами и данными.

* + Мақсаты: Студенттің жадының тиімділігін және шексіз тізбектермен жұмыс істеу қабілетін қоса алғанда, жалқау бағалауды қолданудың артықшылықтарын түсінетінін тексеру.

1. Қатаң және жалқау бағалаудың айырмашылығын түсіндіре аласыз ба?

давайте объясним разницу между строгой (eager) и ленивой (lazy) оценкой.

1. **Строгая оценка (Eager evaluation)**:
   * Строгая оценка означает, что выражение или функция вычисляется сразу, как только оно встречается в коде.
   * В строгой оценке значения выражений вычисляются до того, как они будут использованы в программе.
   * Это означает, что все аргументы функции будут вычислены перед вызовом функции, а значения всех выражений будут вычислены до того, как они будут использоваться.
   * Примеры языков, использующих строгую оценку, включают Python, C++, Java и многие другие.
2. **Ленивая оценка (Lazy evaluation)**:
   * Ленивая оценка означает, что выражение или функция вычисляется только тогда, когда его значение действительно нужно в программе.
   * В ленивой оценке выражения вычисляются только по мере необходимости, что позволяет избежать вычислений, если результат не используется.
   * Ленивая оценка позволяет эффективно работать с бесконечными или потенциально бесконечными структурами данных, так как значения не вычисляются до тех пор, пока они не понадобятся.
   * Примеры языков, поддерживающих ленивую оценку, включают Haskell, Scala, и некоторые части языков, таких как генераторы в Python.

Сравнение:

* В строгой оценке значения вычисляются заранее, по мере их обнаружения в коде. Это может привести к ненужным вычислениям, если результат не используется в программе.
* В ленивой оценке вычисления происходят только по мере необходимости. Это позволяет эффективно использовать ресурсы и работать с большими или потенциально бесконечными данными.

Обе строгая и ленивая оценки имеют свои преимущества и недостатки, и выбор между ними зависит от конкретной задачи и языка программирования, который используется.

* + Мақсаты: Студенттің қатаң (жедел) және жалқау (кешіккен) есептердің айырмашылығын түсінуін бағалау.

1. Жалқау бағалау үшін қандай пайдалану жағдайларын ойлай аласыз?

Ленивая оценка может быть полезной во многих ситуациях, особенно когда имеется дело с большими или бесконечными данными, потенциально занимающими много памяти или времени для вычисления. Вот несколько вариантов использования ленивой оценки:

1. **Обработка потенциально бесконечных последовательностей**: Ленивая оценка позволяет эффективно обрабатывать потенциально бесконечные последовательности данных, такие как потоки данных из файлов или сетевых соединений. Вы можете читать данные из потока по мере необходимости, избегая загрузки всего содержимого в память.
2. **Вычисление больших данных**: При работе с большими объемами данных ленивая оценка может помочь избежать загрузки всех данных в память одновременно. Вы можете обрабатывать данные частями, что позволяет сэкономить память и обрабатывать данные более эффективно.
3. **Генерация последовательностей**: Ленивая оценка позволяет генерировать последовательности чисел или элементов только по мере необходимости. Например, вы можете создать ленивый генератор для генерации бесконечной последовательности простых чисел.
4. **Оптимизация вычислений**: В некоторых случаях ленивая оценка может помочь оптимизировать вычисления, избегая лишних вычислений или пересчетов значений.
5. **Работа с потоками данных**: Ленивая оценка особенно полезна при работе с потоками данных, когда данные поступают по мере их генерации или обработки. Это может быть полезно, например, при обработке данных в реальном времени или при чтении данных из внешних источников.
6. **Вычисление высокоуровневых операций**: Ленивая оценка позволяет создавать высокоуровневые операции, которые могут быть применены к данным по мере необходимости, без необходимости загрузки всех данных сразу. Это может быть полезно, например, при применении фильтров, сортировок или преобразований к данным.
   * Мақсаты: Студенттер жалқау есептеулердің практикалық қолданылуын түсінеді.
7. Жалқау бағалау бағдарламаның жұмысына қалай әсер етуі мүмкін?

Ленивая оценка может повлиять на производительность программы в различных аспектах:

1. **Экономия памяти**: Ленивая оценка позволяет избежать вычислений и создания значений, которые не используются в программе. Это может значительно сэкономить оперативную память, особенно при работе с большими или бесконечными структурами данных. Например, при использовании генераторов в Python, значения вычисляются только по мере необходимости, что позволяет эффективно работать с большими объемами данных.
2. **Улучшение времени выполнения**: Ленивая оценка позволяет избегать лишних вычислений, особенно в случаях, когда не все значения или результаты выражений нужны в программе. Это может привести к сокращению времени выполнения программы, так как необходимо вычислить только те значения, которые действительно используются.
3. **Эффективная работа с бесконечными данными**: Ленивая оценка идеально подходит для работы с бесконечными или потенциально бесконечными данными, так как значения не вычисляются заранее, а создаются по мере необходимости. Например, если вы работаете с потоком данных, ленивая оценка позволит эффективно обрабатывать данные по мере их поступления, без необходимости хранить их все в памяти сразу.
4. **Улучшение параллельных вычислений**: Ленивая оценка может также улучшить производительность в параллельных вычислениях. Поскольку значения вычисляются только по мере необходимости, это позволяет эффективнее использовать ресурсы и распределять задачи между потоками или процессами.
5. **Улучшение модульности и абстракции**: Ленивая оценка позволяет разделять код на более мелкие функции или генераторы, которые могут вычислять значения только по мере необходимости. Это улучшает модульность кода и позволяет создавать более абстрактные и гибкие решения.

В целом, ленивая оценка может значительно повлиять на производительность программы, сокращая потребление памяти, улучшая временные характеристики выполнения и обеспечивая эффективную обработку данных.

* + Мақсаты: Студенттің жалқау бағалаудың нәтижеге әсерін талдау қабілетін бағалау.

1. Python-да қандай генераторлар бар және олардың жалқау бағалауға қалай қатысы барын түсіндіре аласыз ба?

В Python существует несколько типов генераторов, которые связаны с ленивой оценкой и позволяют эффективно работать с данными по мере необходимости. Вот основные типы генераторов:

1. **Генераторы списков (List Comprehensions)**:

Генераторы списков - это компактный и эффективный способ создания списков в Python. Они позволяют создавать списки на основе итерируемых объектов или других списков с помощью синтаксиса в одну строку. Генераторы списков являются ленивыми в том смысле, что они вычисляют значения только по мере необходимости, когда они запрашиваются или используются в программе.

**Генераторы выражений (Generator Expressions)**:

Генераторы выражений - это аналог генераторов списков, но они создают итератор, а не список. Это означает, что они вычисляют значения только по мере необходимости, когда они запрашиваются в программе. Генераторы выражений обычно используются в ситуациях, когда требуется обработка больших объемов данных или когда результат не нужно хранить в памяти полностью.

**Генераторные функции (Generator Functions)**:

Генераторные функции - это функции, которые содержат ключевое слово **yield**, которое возвращает значение из функции и приостанавливает выполнение до следующего вызова. При каждом вызове генераторной функции она продолжает выполнение с того места, где она была приостановлена. Это позволяет создавать генераторы, которые могут производить значения по мере необходимости и не хранить их все в памяти.

Все эти типы генераторов в Python связаны с ленивой оценкой, так как они позволяют вычислять значения по мере необходимости, что помогает экономить память и улучшать производительность программы, особенно при работе с большими объемами данных или бесконечными последовательностями.

* + Мақсаты: Студенттің генераторлар механизмі және олардың жалқау есептеулермен байланысы туралы түсінігін тексеру.

1. Python тілінде функционалды деректер құрылымдарын қалай жүзеге асыра аламыз? Мысалдар келтір.

В Python можно реализовать функциональные структуры данных, такие как списки, множества, словари и другие, используя функциональный подход программирования. В функциональном программировании структуры данных обычно представляются как неизменяемые объекты, а операции над ними создают новые структуры данных, оставляя исходные без изменений. Важно помнить, что функциональный подход к программированию подразумевает работу с неизменяемыми объектами и создание новых структур данных при каждом изменении, вместо модификации существующих.

* + Мақсаты: Студенттер өзгермейтін тізімдер немесе ағаштар сияқты функционалдық деректер құрылымдарын енгізу мен пайдалануды түсінеді.

1. Жаттау дегеніміз не және ол жалқау бағалау жағдайында қалай қолданылады?

Мемоизация - это техника оптимизации, которая заключается в сохранении результатов выполнения функций для предотвращения повторных вычислений при одинаковых входных данных. В контексте ленивой оценки, мемоизация используется для эффективного управления отложенными вычислениями. Когда функция вызывается с определенными аргументами, результат вычисления сохраняется. При последующих вызовах с теми же аргументами результат извлекается из кэша, а не вычисляется снова. Это помогает избежать повторных вычислений и повышает производительность программы.

В случае ленивого вычисления мемоизация может использоваться для управления отложенными вычислениями и оптимизации производительности. Когда результаты вычислений не нужны немедленно, они могут быть сохранены для последующего использования, что позволяет избежать повторных вычислений и улучшает эффективность работы программы. Применение мемоизации в ленивом вычислении позволяет избежать излишних вычислений и оптимизировать использование ресурсов. Например, если у нас есть последовательность значений, которая генерируется лениво, мы можем использовать мемоизацию, чтобы кэшировать результаты вычислений и избежать повторных вычислений для одних и тех же значений.

* + Мақсаты: Оқушылардың есте сақтау техникасы және оны жалқау бағалауды оңтайландыру үшін қолдану туралы білуін қамтамасыз ету.

1. Жалқау бағалаудың қандай кемшіліктері бар?

Хотя ленивая оценка имеет множество преимуществ, она также имеет некоторые недостатки, которые следует учитывать при ее использовании:

1. **Сложность отладки**: Ленивая оценка может усложнить процесс отладки программы из-за того, что вычисления происходят не сразу, а по мере необходимости. Это может привести к тому, что значения переменных и результаты выражений будут неожиданными на первый взгляд.
2. **Непредсказуемое потребление ресурсов**: Иногда сложно предсказать, сколько памяти или процессорного времени потребуется для выполнения программы с использованием ленивой оценки. Некоторые операции могут потреблять больше ресурсов, чем ожидалось из-за ленивого вычисления.
3. **Скрытые зависимости и побочные эффекты**: Ленивая оценка может привести к скрытым зависимостям и побочным эффектам, особенно если функции изменяют глобальное состояние или зависят от порядка вычислений. Это может сделать программу сложнее для понимания и поддержки.
4. **Низкая предсказуемость производительности**: Поскольку вычисления происходят по мере необходимости, производительность программы с использованием ленивой оценки может быть менее предсказуемой, чем у программы с энергичной оценкой. Это может быть особенно проблематично при работе с реальным временем или при необходимости точного контроля над производительностью.
5. **Потенциальные утечки памяти**: В некоторых случаях ленивая оценка может привести к утечкам памяти из-за того, что значения не освобождаются сразу после использования. Это особенно верно в случаях, когда генерируются бесконечные структуры данных или когда результаты вычислений сохраняются на протяжении всего выполнения программы.

Хотя ленивая оценка может быть мощным инструментом для оптимизации производительности и управления ресурсами, важно учитывать эти недостатки и принимать меры для их минимизации при проектировании и разработке программ.

* + Мақсаты: Студенттердің сыни ойлауын және жалқау есептеулердің ықтимал шектеулері туралы хабардарлығын бағалау.

1. Жалқау бағалауды қолданатын нақты өмірлік мәселеге мысал келтіріңіз.

Одним из распространенных примеров, когда ленивая оценка может быть полезна, является обработка и анализ больших объемов данных, особенно когда часть данных может быть избыточна или ненужна для текущей задачи. Рассмотрим ситуацию, когда у нас есть набор данных, содержащий информацию о пользователях социальной сети, включая их профили, списки друзей, посты и т. д.

Проблема: Предположим, что мы хотим проанализировать активность пользователей в социальной сети и определить тех пользователей, которые имеют больше всего активности в виде лайков, комментариев и публикаций.

Решение без использования ленивой оценки: Если мы загружаем все данные сразу и затем выполняем анализ, это может привести к ненужному расходу памяти и времени на загрузку всех данных, даже если нам нужна только небольшая часть информации для выполнения нашей задачи. Например, если у нас есть миллионы пользователей, а мы интересуемся только топ-100 самых активных, загрузка всех данных может быть избыточной.

Решение с использованием ленивой оценки: Вместо загрузки всех данных сразу мы можем использовать ленивую оценку для загрузки данных по мере необходимости. Например, мы можем создать генератор, который будет загружать профили пользователей по одному и анализировать их активность. Когда мы достигнем топ-100 пользователей с наибольшей активностью, мы можем остановить процесс загрузки дополнительных данных.

* + Мақсаты: Студенттің теориялық білімін практикада қолдана білу қабілетін тексеру.

# Примерная Задача для Лабораторной Работы 5

Задача: Создание и Использование Генератора для Генерации Бесконечной Арифметической Последовательности

Ленивые вычисления в Python часто реализуются с помощью генераторов. Генераторы позволяют вычислять и возвращать значения по одному, по мере необходимости, вместо вычисления всей последовательности сразу. В этой задаче необходимо создать генератор, который будет производить бесконечную арифметическую последовательность (например, натуральные числа).

Цель задачи:

Написать функцию-генератор, которая начинает с заданного числа и продолжает генерировать элементы арифметической последовательности с заданным шагом.

Решение:

def infinite\_arithmetic\_sequence(start, step):

"""

Генератор бесконечной арифметической последовательности. start - начальное значение последовательности

step - шаг последовательности """

current = start while True:

yield current current += step

# Пример использования генератора

sequence = infinite\_arithmetic\_sequence(1, 1) # Начальное число 1, шаг 1

# Получение первых 10 элементов последовательности for \_ in range(10):

print(next(sequence)) Объяснение:

Функция `infinite\_arithmetic\_sequence` является генератором, который начинает с числа `start` и бесконечно увеличивает его на `step` при каждом вызове `next()`. Ключевое слово `yield` используется для возврата следующего значения в последовательности и приостановки выполнения функции до следующего вызова

`next()`.

Это решение демонстрирует использование ленивых вычислений в Python. Вместо того чтобы создавать и хранить всю последовательность в памяти, генератор вычисляет каждое следующее значение по мере необходимости. Это делает подход с генераторами особенно полезным для работы с большими данными или потенциально бесконечными последовательностями.