**Лабораторная Работа 5: Ленивые Вычисления и Функциональные**

**Структуры Данных**

**Цель:**

Изучение и применение концепций ленивых вычислений и функциональных структур данных в Python. Целью лабораторной работы является понимание принципов ленивого выполнения кода, ознакомление со специфическими функциональными структурами данных и развитие навыков их эффективного использования в реальных задачах программирования.

Задачи:

1. Понимание Ленивых Вычислений:

* Изучение концепции ленивых вычислений и её отличий от строгой (энергичной) оценки.
* Рассмотрение примеров ленивых вычислений в Python, таких как генераторы и выражения-генераторы.

2. Применение Ленивых Вычислений в Практических Задачах:

* Разработка решений для задач, где ленивые вычисления могут повысить эффективность, например, в задачах обработки больших объёмов данных.
* Исследование влияния ленивых вычислений на производительность и потребление памяти.

3. Освоение Функциональных Структур Данных:

* Изучение функциональных аналогов традиционных структур данных, таких как неизменяемые списки, деревья и карты.
* Разработка умения использовать и создавать неизменяемые структуры данных в Python.

4. Реализация Примеров с Ленивыми Вычислениями и Функциональными Структурами:

* Создание примеров и решений задач, демонстрирующих использование ленивых вычислений и функциональных структур данных.
* Анализ преимуществ и ограничений таких методов в различных контекстах программирования.

5. Развитие Навыков Аналитического Мышления и Проблемного Решения:

* Поощрение самостоятельного исследования и экспериментирования с новыми подходами в программировании.
* Развитие способности критически оценивать и выбирать подходящие инструменты и методы для конкретных задач.

Важность Лабораторной Работы:

Эта лабораторная работа предназначена для расширения понимания студентов о возможностях и преимуществах функционального программирования, включая ленивые вычисления и функциональные структуры данных. Она направлена на развитие навыков эффективной работы с большими объёмами данных и создания более чистого, модульного кода. Кроме того, работа способствует улучшению аналитических навыков и гибкости мышления при решении программных задач.

Функциональное Декомпозирование

* + Создать серию взаимосвязанных ленивых функций, каждая из которых выполняет одно действие в цепочке обработки данных.

def lazy\_load\_data(source):  
 print("Loading data from:", source)  
 return [1, 2, 3, 4, 5]  
def lazy\_preprocess\_data(data):  
 print("Preprocessing data")  
 return [x \* 2 for x in data]  
def lazy\_filter\_data(data):  
 print("Filtering data")  
 return [x for x in data if x > 3]  
def lazy\_transform\_data(data):  
 print("Transforming data")  
 return [x \*\* 2 for x in data]  
def lazy\_save\_data(data, destination):  
 print("Saving data to:", destination)  
 # Here you can implement saving logic  
 print("Data saved successfully")  
def data\_processing\_pipeline(source, destination):  
 data = lazy\_load\_data(source)  
 data = lazy\_preprocess\_data(data)  
 data = lazy\_filter\_data(data)  
 data = lazy\_transform\_data(data)  
 lazy\_save\_data(data, destination)  
data\_processing\_pipeline("input\_data.txt", "output\_data.txt")

Мысалы, банкоматты пайдалану туралы. процесті келесіге бөлу:

1. Банкоматқа дейін жаяу бару
2. Банк картаңызды салу
3. PIN кодын енгізу

Бағдарламалауды дәл осылай ойлауға болады. Осы банкоматты басқаратын бағдарламалық құралды ойлап көреміз:

1. Картаны оқуға арналған код
2. PIN кодын растау
3. Тасымалдау өңдеу

Олардың әрқайсысын одан әрі бөлуге болады. Ішкі жүйенің ең ыдырайтын бөліктеріне жеткеннен кейін, сол бөліктерді кодтауды қалай бастау керектігі туралы ойлануға болады. Содан кейін сіз сол шағын бөліктерді үлкен бүтінге құрасыз. Осы

Википедия:

[Декомпозиция (бағдарламалау)](http://en.wikipedia.org/wiki/Decomposition_(computer_science))

Функционалдық декомпозицияның артықшылығы мынада: кодтауды бастағаннан кейін сіз қолданбаңыз үшін жұмыс істеуге болатын ең қарапайым компоненттермен жұмыс жасайсыз. Сондықтан бұл құрамдастарды әзірлеу және сынау әлдеқайда жеңіл болады (сіздің қажеттіліктеріңізге сәйкес код пен жобаны жақсырақ құрастыра алатыныңызды айтпағанда).

Айқын кемшілігі - уақытты инвестициялау. Күрделі жүйеде функционалдық ыдырауды орындау үшін кодтау басталғанға дейін тривиальды уақыттан астам уақыт қажет.

Бұл WorkBreakDown Structures (WBS), mindMapping және жоғарыдан төмен әзірлеумен бірдей - негізінен үлкен мәселені кішірек, түсінікті ішкі бөліктерге бөлу.

Артықшылықтары

* бағдарламалауға проактивті көзқарасқа мүмкіндік береді (кодқа деген ынтадан бас тарту)
* жобаның күрделі және/немесе тәуекел аймақтарын анықтауға көмектеседі (банкомат мысалында қауіпсіздік күрделірек құрамдас болуы мүмкін)
* жобаның БАРЛЫҚ құрамдас бөліктерін анықтауға көмектеседі - жоба/код сәтсіздігінің №1 себебі (Каперс Джонс арқылы) бөліктердің жетіспеуі - жобаның соңына дейін ойланбаған нәрселер (жоқ, мен адамның балансын тексеру керек екенін түсінбедім. $ бергенге дейін)
* жақсырақ бағдарламалау, кодты ортақ пайдалану және жұмысты тарату үшін компоненттерді ажыратуға мүмкіндік береді

Жаман жақтары - ыдырауды орындауда нақты ЖҰМЫСТАР жоқ, дегенмен кейбір жиі кездесетін қателер бар

* жеткілікті түрде үзілмеу немесе алысқа дейін бөлшектемеу - әр адам құрамдас бөлік туралы түсінікті қамтамасыз ету үшін қажетті егжей-тегжейлі деңгейді анықтауы керек (кодтың бағдарламалау жолдарына бөлінбеңіз...)
* бұрыннан бар үлгілерді/код модульдерін ескермеу (қайта өңдеу)
* көлемінің дұрыстығына көз жеткізу үшін клиенттермен тексермеу
* нақты кодтау кезінде бұзылуды пайдаланбау (мысалы, жоспарды ұмытып, кейбір тақталарды біріктіре бастағаннан гөрі үйді жобалау)
* # Определение констант для операций  
  DEPOSIT = "DEPOSIT"  
  WITHDRAW = "WITHDRAW"  
  DISPLAY = "DISPLAY"  
    
  def check\_pin(input\_pin, card\_pin):  
   return input\_pin == card\_pin  
    
  def deposit\_money(card\_balance):  
   money = float(input("Enter the sum to DEPOSIT: "))  
   card\_balance += money  
   print("Deposited: $", money)  
   print("Current balance:", card\_balance)  
   return card\_balance  
    
  def withdraw\_money(card\_balance):  
   money = float(input("Enter the sum to WITHDRAW: "))  
   if money <= card\_balance:  
   card\_balance -= money  
   print("Withdrawn: $", money)  
   else:  
   print("Insufficient funds!")  
   print("Current balance:", card\_balance)  
   return card\_balance  
    
  def display\_balance(card\_balance):  
   print("Current balance:", card\_balance)  
    
  def perform\_action(action, card\_balance):  
   if action == DEPOSIT:  
   card\_balance = deposit\_money(card\_balance)  
   elif action == WITHDRAW:  
   card\_balance = withdraw\_money(card\_balance)  
   elif action == DISPLAY:  
   display\_balance(card\_balance)  
   else:  
   print("Invalid action!")  
    
  card\_number = input("Enter card number: ")  
  input\_pin = input("Enter PIN: ")  
  card\_pin = "1234"  
  card\_balance = 1000  
    
  if check\_pin(input\_pin, card\_pin):  
   action = input("Enter desired action: ")  
   perform\_action(action, card\_balance)  
  else:  
   print("Incorrect pin!")
* **Python функционалдық ыдырауы дегеніміз не?**
* **Функционалды декомпозиция** күрделі мәселені деректер қатынастарына емес, орындалуы қажет тапсырмаларға негізделген қарапайым мәселелерге бөлу тәсілі болып табылады. Бұл термин әдетте ескі процедураға бағытталған дизайнмен байланысты.
* **Функционалдық ыдырау дегеніміз не?**
* **Функционалды декомпозиция - инженерлер құрылғының, жүйенің немесе процестің жалпы функциясын оның кішірек бөліктеріне бөлетін қадамдар жиынтығын** сипаттау үшін қолданатын термин . ... Функция - бұл құрылғы, жүйе немесе процесс орындайтын жай ғана тапсырма. Ыдырау - ыдырау процесі.
* **Python-да ыдырау дегеніміз не?**
* Автордың суреті. Уақыт қатарларының декомпозициясы - уақыттық қатарды әрқайсысы негізгі үлгі санатын, трендті, маусымдылықты және шуды білдіретін **бірнеше құрамдас бөліктерге бөлетін әдіс** . Бұл оқулықта біз сізге Python көмегімен уақыт қатарын қалай автоматты түрде бөлшектеу керектігін көрсетеміз.
* **Функционалды декомпозицияны қандай программалау тілі жүзеге асырады?**
* Функционалдық декомпозиция деректерді өңдеу мақсатында дискретті функцияларға негізделген, мысалы, **Джексон құрылымдық бағдарламалауын** пайдалану . Функциялар көбінесе объектіге бағытталған ортадағы әдістер болып табылады.
* **Функционалдық ыдырау диаграммасы дегеніміз не?**
* Функционалдық декомпозиция диаграммасы **бүкіл функцияны немесе жобаны және оны орындау үшін қажетті барлық қажетті қосалқы тапсырмаларды** қамтиды . Функционалды декомпозиция бизнес пен өнеркәсіптен бастап компьютерлік бағдарламалау мен AI-ға дейін бірнеше контексте қолданылатын мәселелерді шешу құралы болып табылады.
* **Неліктен функционалдық ыдырау нашар?**
* Функционалдық ыдырау **жүйенің функционалдығы негізінде оның құрылымдық блоктарына жүйені ыдыратады** . Мысалы, жүйе шот-фактура, есепшот және жөнелтілім сияқты операциялар жинағын орындау қажет болса, сіз Шот-фактура қызметі, Есепшот ұсыну қызметі және Жеткізу қызметімен аяқталасыз.
* **Бірінші деңгейлі функционалдық ыдырау дегеніміз не?**
* Негізгі идея Функционалды декомпозиция үлкенірек тапсырманы қосалқы тапсырмаларға бөлуді қамтиды. ... 1-сурет: Функционалдық ыдыраудың бір деңгейі. Функционалдық декомпозицияның мақсаты **күрделі тапсырманы** түсінуге оңай, содан кейін қол жеткізудің практикалық жолдарын табу оңайырақ кішігірім тапсырмалар тізбегіне айналдыру болып табылады.
* **Уақыт қатарларының ыдырауы қалай аталады?**
* Бұл **детрендинг** деп аталады . Уақыт қатарларының деректері жиі бірнеше құрамдас бөліктерден тұрады деп есептеледі: ұзақ мерзімді тренд, маусымдық өзгерістер және тұрақты емес вариациялар.
* **Сіз ыдырауды қалай болжаасыз?**
* Декомпозиция үлгісін пайдаланып уақыт қатарын болжау үшін **әрбір бөлек құрамдас үшін болашақ мәндерді есептеп, болжамды алу үшін оларды қайтадан қосасыз** . Содан кейін мәселе компоненттердің әрқайсысы үшін ең жақсы үлгіні табу болып табылады.

## 1Неліктен функционалдық ыдырау қолданылады?

Функционалды декомпозиция функционалдық талаптарды тиімдірек анықтауға, ұйымдастыруға және хабарлауға көмектеседі. Жүйені басқарылатын бірліктерге бөлу арқылы құрамдас бөліктер арасындағы тәуелділік пен өзара әрекеттесуді анықтау кезінде күрделілік пен анық еместікті азайтуға болады. Оған қоса, ол құрамдастарды әртүрлі топтарға немесе фазаларға басымдық беруге және бөлуге, сондай-ақ жүйе мақсаттары мен пайдаланушы қажеттіліктеріне қатысты талаптарды тексеруге және растауға көмектеседі. Оған қоса, ол болашақ өзгерістер немесе жақсартулар үшін құрамдастарды қайта пайдалануға және өзгертуге мүмкіндік береді.

## 2Функционалдық ыдырау қалай орындалады?

Функционалдық ыдырауға бір өлшемді тәсіл жоқ, бірақ процесті басқару үшін кейбір жалпы қадамдарды орындауға болады. Жүйенің ауқымы мен мақсаттарын анықтаудан және жоғары деңгейлі функцияларды анықтаудан бастаңыз. Содан кейін, функциялардың, ішкі функциялардың және тапсырмалардың иерархиясын жасау үшін жоғарыдан төмен қарай әдісті пайдаланыңыз. Мұны диаграмма, кесте немесе тізім арқылы көрсетуге болады. Мысалы, пайдаланушыны тіркеген кезде пайдаланушы енгізуін растау (мысалы, электрондық пошта пішімін және құпия сөз күшін тексеру), пайдаланушы тіркелгісін жасау (мысалы, пайдаланушы идентификаторын жасау және пайдаланушы деректерін сақтау) және пайдаланушыға кіру (мысалы, тіркелгі деректерінің түпнұсқалығын растау және профильді көрсету). Бұл процесс жобаңызға жеткілікті егжей-тегжейлі деңгейге жеткенше қайталануы керек.

## 3Функционалдық декомпозицияны қалай құжаттау керек?

Функциялардың, ішкі функциялардың және тапсырмалардың иерархиясын жасағаннан кейін оларды анық және дәйекті түрде құжаттау керек. Функционалдық декомпозицияны құжаттау үшін әртүрлі пішімдерді және құралдарды пайдалануға болады, мысалы, функционалдық ыдырау диаграммасы, ол компоненттер арасындағы байланысты көрсету үшін қораптар мен сызықтармен иерархияның графикалық көрінісі болып табылады. Функционалды декомпозиция матрицасы – әрбір құрамдастың деңгейі мен атын көрсету үшін жолдар мен бағандар бар иерархияның кестелік көрінісі. Сонымен қатар, функционалдық спецификация құжаты әрбір құрамдастың мақсаты мен көлемін сипаттайтын тақырыптар мен параграфтары бар иерархияның мәтіндік көрінісі болып табылады. Бұл құжаттарды жасау және жаңарту үшін Visio немесе Lucidchart сияқты бағдарламалық құралдарды, Excel немесе Google Sheets сияқты электрондық кесте құралдарын және Word немесе Google Docs сияқты мәтіндік процессор құралдарын пайдалануға болады

## 4Функционалдық ыдырауды қалай тексеруге болады?

Функционалдық декомпозицияның соңғы қадамы нәтижелерді тексеру және олардың жүйе мақсаттары мен пайдаланушы қажеттіліктеріне сәйкес келуін қамтамасыз ету болып табылады. Мұны істеу үшін тексеру тізімі, сауалнама немесе мүдделі тараптармен кездесу сияқты шолу және кері байланыс әдістерін пайдалануға болады. Тестілеу және прототиптеу функционалдық ыдырау мақсаты бойынша жұмыс істейтінін және сапа стандарттарына сәйкес келетінін тексеру үшін де маңызды. Сонымен қатар, қадағалау және теңестіру функционалдық ыдырауды басқа талаптармен және дизайн артефактілерімен, мысалы, пайдаланушы оқиғалары немесе пайдалану жағдайлары, қадағалау матрицасын немесе айқас сілтеме жасау әдісін пайдаланып қадағалау арқылы орнатылуы керек.

## 5Міне, тағы не ескеру керек

Бұл алдыңғы бөлімдердің ешқайсысына сәйкес келмейтін мысалдарды, оқиғаларды немесе түсініктерді бөлісуге арналған кеңістік. Тағы не қосқыңыз келеді?

## 1. Шолу

Бұл оқулықта **біз функционалдық ыдырауды практикалық мысалмен егжей-тегжейлі талқылаймыз.**

Сонымен қатар, біз бұл әдістің артықшылықтары мен кемшіліктерін қарастырамыз.

## 2. Кіріспе

**Функционалдық декомпозиция - күрделі жүйені кішірек, басқарылатын құрамдастарға бөлуді қамтитын жүйені жобалау және талдау әдісі.** Сонымен қатар, біз нақты функциялар мен қатынастарды пайдалана отырып, күрделі жүйелерді бұзамыз.

Бұл жүйені оның құрамдас бөліктеріне бөлу, функцияларды анықтау және жүйенің мақсаттарына жету үшін олардың өзара әрекеттесуін анықтау процесі. Функционалдық декомпозицияның мақсаты күрделі жүйелерді кішірек, басқарылатын және түсінікті бөліктерге оңайлату болып табылады. Функционалдық декомпозиция күрделі жүйелерді ыңғайлы түрде жобалауды, талдауды және енгізуді жеңілдетеді.

[Бағдарламалық жасақтаманы](https://en.wikipedia.org/wiki/Software_engineering) , [жүйелік инженерияны](https://en.wikipedia.org/wiki/Systems_engineering) , бизнес-процестерді жетілдіруді және [робототехниканы](https://en.wikipedia.org/wiki/Robotics) қоса алғанда, әртүрлі салаларда кең ауқымды қосымшаларға ие .

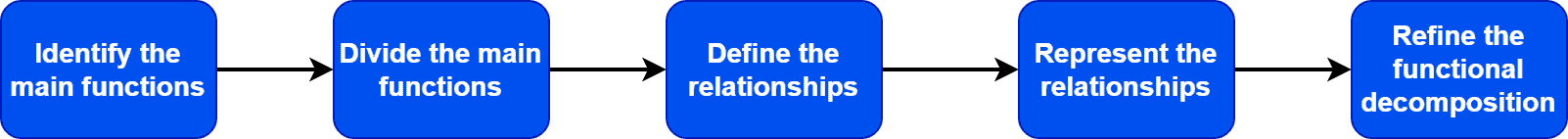
Бағдарламалық жасақтамада біз функционалдық декомпозицияны қолдана отырып, күрделі бағдарламалық жүйелерді жеңілдетеміз. Сонымен қатар, біз әзірлеуді қажет ететін әртүрлі компоненттер мен функцияларды анықтай аламыз. Сонымен қатар, күрделі жүйелерді жобалау үшін жүйелік инженерия талаптарды талдау үшін функционалдық ыдырау әдістерін қолданады.

Сондай-ақ, біз бұл әдісті бизнес-процестерді кішірек құрамдас бөліктерге бөлу және жақсарту бағыттарын анықтау үшін әрқайсысын талдау арқылы оңтайландыру үшін пайдалана аламыз. Сонымен қатар, біз оны роботтардың әртүрлі компоненттерін, соның ішінде сенсорларды, жетектерді және [басқару жүйелерін](https://en.wikipedia.org/wiki/Control_system) жобалау және зерттеу үшін пайдалана аламыз .

Функционалды декомпозиция - күрделі жүйелердің кең ауқымына қолдануға болатын және жобалауды, талдауды және енгізу процесін жеңілдетуге көмектесетін әмбебап әдіс.

## 3. Қадамдар

Функционалды ыдырау күрделі жүйені нақты функцияларына негізделген кішірек, қарапайым компоненттерге бөледі. Бұл процесс әдетте бес қадамнан тұрады:



**Бірінші қадам - ​​жүйенің негізгі функцияларын анықтау.** Бұл процесс жүйенің не істейтінін және ол неге қол жеткізуі керек екенін анықтауды қамтиды. Келесі қадам негізгі функцияларды кішірек қосалқы функцияларға бөлу болып табылады. Оған қоса, әрбір негізгі функция нақты тапсырмаларына байланысты кішірек, басқарылатын бөліктерге бөлінеді.

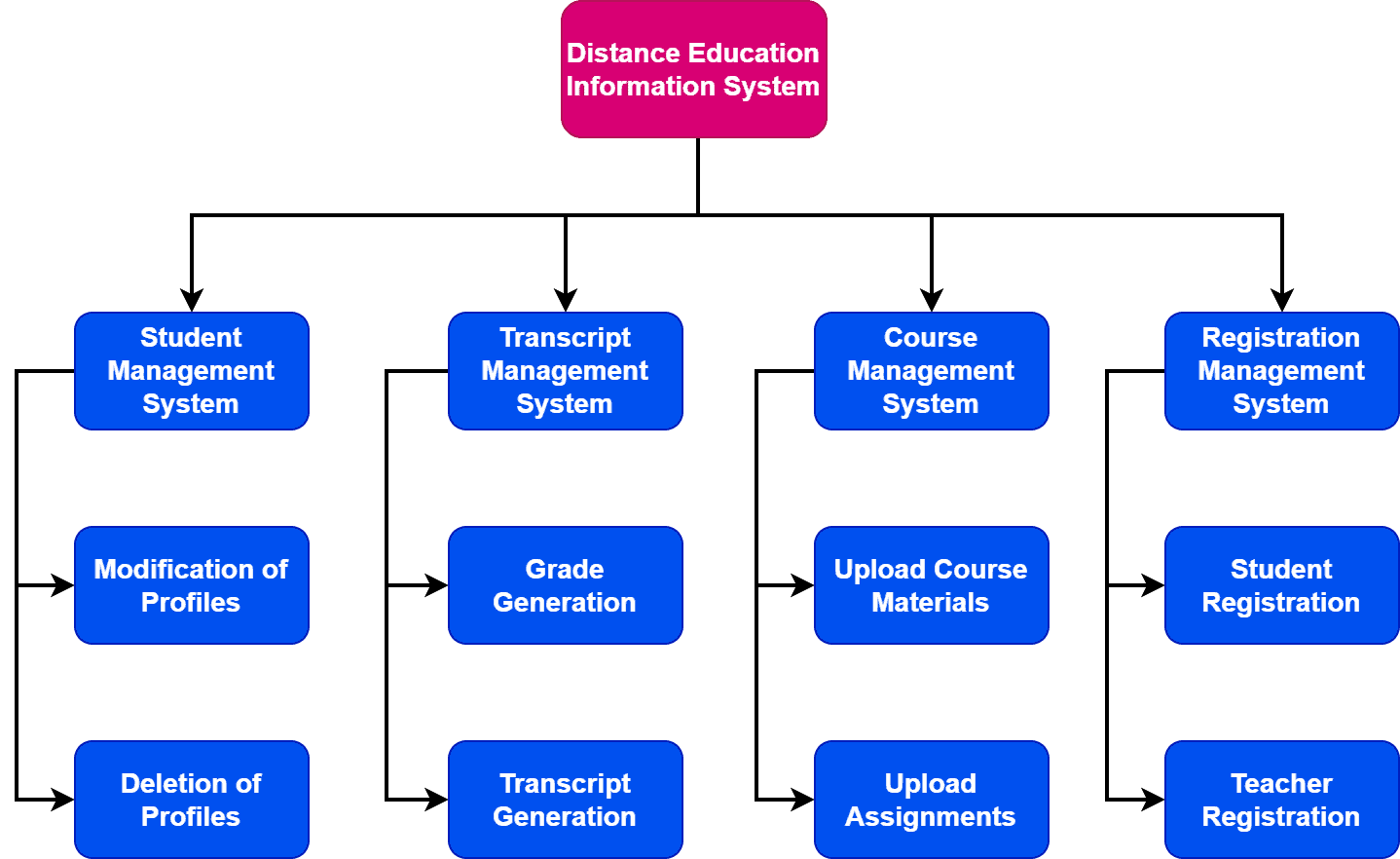
Бұдан басқа, **біз ішкі функциялар арасындағы байланыстарды анықтаймыз.** Бұл жүйенің жалпы мақсаттарына жету үшін ішкі функциялардың өзара әрекеттесетінін және бірлесіп жұмыс істейтінін түсінуді қамтиды. Алға қарай біз ішкі функциялар арасындағы қарым-қатынастарды көрнекі түрде көрсетеміз. Біз оны әдетте әртүрлі ішкі функцияларды және олардың өзара әрекеттесуін көрсететін [функционалды блок-схема арқылы жасаймыз.](https://en.wikipedia.org/wiki/Functional_block_diagram)

Соңында біз функционалдық ыдырауды нақтылаймыз. Ол функционалды блок-схеманы қарауды қамтиды. Оған қоса, біз жүйені және оның функцияларын дәл көрсету үшін қажетті өзгертулер енгіземіз.

Күрделі жүйені кішірек, қарапайым құрамдас бөліктерге бөлу арқылы функционалдық ыдырау жобалау мен талдау процесін жеңілдетуге көмектеседі, оны түсіну мен жүзеге асыруды жеңілдетеді.

## 4. Мысал

Күрделі жүйені жеңілдету үшін функционалдық декомпозицияны қолданатын практикалық мысалды талқылайық. Мұнда **студенттер барлық курстарды онлайн режимінде басқара және қол жеткізе алатын қашықтықтан білім беру ақпараттық жүйесінің мысалын ұсынамыз.** Қашықтықтан білім берудің ақпараттық жүйесін бұзу үшін функционалдық декомпозицияны қалай қолдануға болатынын қарастырайық:



Мұнда қашықтан білім берудің ақпараттық жүйесін басқаруға оңай төрт бөлікке бөлеміз: студенттерді басқару, транскрипттерді басқару, курстарды басқару және тіркеуді басқару жүйелері. Сонымен қатар, біз әрбір ішкі жүйені екі бөлікке бөлеміз.

Мысалы, студенттер профильдерін өзгерте алады және студенттерді басқару жүйесі бойынша кез келген қажетті ақпаратты қоса алады. Сонымен қатар, жүйелік әкімші білім алуын аяқтаған кезде оның профилін жоя алады. Осылайша, функционалдық ыдырау күрделі жүйені басқарылатын ішкі жүйелерге бөледі.

## 5. Артықшылықтары

Функционалдық ыдырау бірнеше артықшылықтарға ие. Оның кейбір маңызды артықшылықтарын талқылайық.

**Ең маңызды артықшылығы - бұл күрделі жүйелерді жеңілдетеді.** Күрделі жүйені кішірек, қарапайым құрамдас бөліктерге бөлу арқылы функционалдық ыдырау жүйені түсінуді және талдауды жеңілдетеді. Бұған қоса, бұл ынтымақтастықты жақсартады. Әртүрлі құрамдас бөліктер арасындағы қарым-қатынастарды анықтау арқылы функционалдық декомпозиция әртүрлі командалар мен мүдделі тараптарға жалпы жүйе мақсаттарына жету үшін қалай бірге жұмыс істеу керектігін түсінуге көмектеседі.

**Ол жүйелерде қайта пайдалануға мүмкіндік береді.** Жүйені кішірек, модульдік құрамдас бөліктерге бөлетіндіктен, функционалдық ыдырау болашақ жобаларда құрамдастарды қайта пайдалануды жеңілдетеді. Демек, бұл жүйе дизайнерлері мен инженерлерінің уақыты мен ресурстарын үнемдейді.

Ақырында, **бұл мүдделі тараптар арасындағы байланысты жақсартуға көмектеседі.** Әртүрлі құрамдас бөліктер мен қарым-қатынастардың анық көрнекі көрінісін қамтамасыз ете отырып, ол мүдделі тараптар арасындағы байланысты жақсартады. Сонымен қатар, бұл түсінбеушіліктерді болдырмауға көмектеседі.

Тұтастай алғанда, функционалдық декомпозиция күрделі жүйелерді жеңілдетуге, ынтымақтастықты жақсартуға және жобалауды, талдауды және енгізуді жеңілдетуге құрылымдық және жүйелі тәсілді қамтамасыз етеді.

## 6. Кемшіліктері

**Функционалдық декомпозиция, кез келген басқа әдіс сияқты, кейбір кемшіліктерге ие, соның ішінде уақытты тұтыну, икемділікте шектеу, тым жеңілдету қаупі, интеграцияның болмауы және домендік сараптамаға тәуелділік.**

Күрделі жүйені кішірек құрамдас бөліктерге бөлу және олардың өзара байланысын анықтау процесі көп уақытты қажет етеді. Сонымен қатар, бұл айтарлықтай күш-жігерді қажет етеді. Сонымен қатар, функционалдық ыдырау орындалғаннан кейін, процесті қайта бастамай-ақ жүйеге өзгертулер енгізу қиын болуы мүмкін.

Функционалдық ыдыраудың тағы бір кемшілігі - артық жеңілдету қаупі. Күрделі жүйені кішірек құрамдас бөліктерге бөлу арқылы жүйені тым жеңілдету және оның күрделілігін дәл көрсетпеу қаупі бар. Сонымен қатар, функционалдық ыдырау жүйенің әртүрлі бөліктері арасындағы өзара әрекеттесулер мен қатынастарды елемеуге болады, өйткені ол жеке құрамдастарға назар аударады, бұл интеграцияның болмауына әкеледі.

Ақырында, функционалдық декомпозицияның сәттілігі талдауды жүзеге асыратын адамдардың домендік сараптамасына байланысты. Сонымен қатар, нәтижелер олардың білімі мен тәжірибесімен шектелуі мүмкін.

Бұл шектеулерді ескеру қажет болғанымен, функционалдық ыдырау күрделі жүйелерді жеңілдету және жобалауды, талдауды және енгізуді жеңілдету үшін пайдалы құрал болып қала береді.

## 7. Қорытынды

Бұл оқулықта біз практикалық мысалмен функционалдық ыдырау қалай жұмыс істейтінін талқыладық. Сонымен қатар, біз бұл әдістің негізгі артықшылықтары мен кемшіліктерін зерттедік.

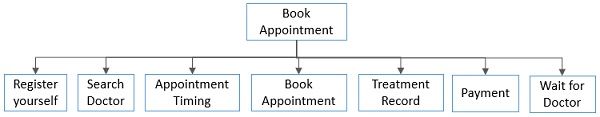
Функционалды ыдырау – күрделі процесті оның жеке элементтерін зерттеу мақсатында бөлшектейтін талдау әдісі. Функционалды декомпозиция бизнес операцияларын, компьютерлік бағдарламалауды, машиналық оқытуды және басқа да көптеген салалардағы мәселелерді шешуге және көмек көрсетуге көмектеседі.

## Функционалдық ыдырау дегеніміз не?

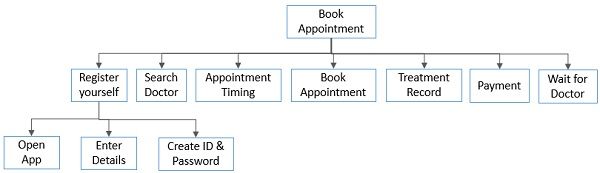
Функционалдық декомпозиция кең мағынада функционалдық қатынасты оның құрамдас бөліктеріне бастапқы функцияны сол бөліктерден қайта құруға болатындай етіп шешу процесін білдіреді.

## Компьютерлік бағдарламалаудағы декомпозиция дегеніміз не?

Декомпозиция информатиканың төрт негізінің бірі болып табылады. Ол күрделі мәселені немесе жүйені басқаруға болатын және түсіну оңайырақ кішірек бөліктерге бөлуді қамтиды. Одан кейін кішірек бөлшектерді зерттеуге және шешуге немесе жеке жобалауға болады, өйткені олармен жұмыс істеу оңайырақ.



1. register yourself
2. search for doctor
3. manage the appointment timing
4. book appointment
5. treatment record
6. payment
7. wait for doctor



Функционалды ыдырау

* + Әрқайсысы деректерді өңдеу тізбегінде бір әрекетті орындайтын өзара байланысты жалқау функциялар қатарын жасаңыз.

Functional Decomposition/Disintegration

* + Create a series of interconnected lazy functions, each of which performs one action in the data processing chain.

Funktsional Parchalanish

* + Har biri ma'lumotlarni qayta ishlash zanjirida bitta amalni bajaradigan o'zaro bog'langan dangasa funktsiyalar qatorini yarating.

Эти задачи призваны помочь студентам понять и применить концепции ленивых вычислений и функционального программирования, а также научить их создавать и использовать функциональные структуры данных в Python.

**Критерии Оценки:**

* Написание кода для решения индивидуальной задачи: 1 балл
* Объяснение и понимание написанного кода во время защиты: 2 балла - Ответ на один из теоретических вопросов, выбранный преподавателем: 1 балл

**Вопросы для Подготовки:**

1. Что понимается под ленивыми вычислениями в Python и как они реализуются?

- Цель: Убедиться, что студент понимает концепцию ленивых вычислений и способы их реализации в Python.

2. Каковы преимущества ленивых вычислений?

- Цель: Проверить, осознает ли студент преимущества использования ленивых вычислений, включая эффективность использования памяти и возможность работы с бесконечными последовательностями.

3. Можете ли вы объяснить разницу между строгими и ленивыми вычислениями?

- Цель: Оценить понимание студентом различий между строгими (немедленными) и ленивыми (отложенными) вычислениями.

4. Какие сценарии использования ленивых вычислений вы можете привести?

* Цель: Понимание студентами практических применений ленивых вычислений. 5. Как ленивые вычисления могут повлиять на производительность программы?
* Цель: Оценка способности студента анализировать влияние ленивых вычислений на производительность.

6. Можете ли вы объяснить, что такое генераторы в Python и как они связаны с ленивыми вычислениями?

- Цель: Проверка понимания студентом механизма генераторов и их отношения к ленивым вычислениям.

7. Как можно реализовать функциональные структуры данных в Python? Приведите примеры.

- Цель: Понимание студентами реализации и использования функциональных структур данных, таких как неизменяемые списки или деревья.

8. Что такое мемоизация и как она применяется в контексте ленивых вычислений?

- Цель: Убедиться, что студенты осведомлены о технике мемоизации и её применении для оптимизации ленивых вычислений.

9. Какие недостатки могут быть у ленивых вычислений?

- Цель: Оценить критическое мышление студентов и осознание потенциальных ограничений ленивых вычислений.

10. Приведите пример реальной задачи, где вы бы применили ленивые вычисления.

- Цель: Проверить способность студента применять теоретические знания на практике.

Эти вопросы нацелены на оценку уровня понимания студентами концепции ленивых вычислений, их умения работать с функциональными структурами данных и использовать эти концепции для решения реальных задач.

**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.

5-зертхана: Жалқау есептеу 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.

Функционалдық ыдырау

- Әрқайсысы деректерді өңдеу тізбегінде бір әрекетті орындайтын өзара байланысты жалқау функциялар қатарын жасаңыз

Бұл есептер студенттерге жалқау бағалау және функционалдық бағдарламалау ұғымдарын түсінуге және қолдануға көмектесу және оларды Python тілінде функционалды деректер құрылымдарын құру және пайдалануды үйрету үшін жасалған.

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.

2. Жалқау бағалаудың қандай артықшылықтары бар?

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

3. Қатаң және жалқау бағалаудың айырмашылығын түсіндіре аласыз ба?

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

4. Жалқау бағалау үшін қандай пайдалану жағдайларын ойлай аласыз?

- Мақсаты: Студенттер жалқау есептеулердің практикалық қолданылуын түсінеді. 5. Жалқау бағалау бағдарламаның жұмысына қалай әсер етуі мүмкін?

- Мақсаты: Студенттің жалқау бағалаудың нәтижеге әсерін талдау қабілетін бағалау.

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

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

7. Python тілінде функционалды деректер құрылымдарын қалай жүзеге асыра аламыз? Мысалдар келтіріңіз.

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

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

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

9. Жалқау бағалаудың қандай кемшіліктері бар?

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

10. Жалқау бағалауды қолданатын нақты өмірлік мәселеге мысал келтіріңіз.

- Мақсаты: Студенттің теориялық білімін практикада қолдана білу қабілетін тексеру.

Бұл сұрақтар студенттердің жалқау бағалау түсінігін түсінуін, олардың функционалдық деректер құрылымдарымен жұмыс істеу қабілетін және осы ұғымдарды нақты дүниелік мәселелерді шешу үшін пайдалана білуін бағалауға бағытталған.

5-зертханаға арналған есеп үлгісі

Тапсырма: Шексіз арифметикалық тізбекті құру үшін генераторды құру және пайдалану

Python-да жалқау бағалау көбінесе генераторлар арқылы жүзеге асырылады. Генераторлар барлық тізбекті бірден бағалаудың орнына мәндерді қажетінше бір-бірден бағалауға және қайтаруға мүмкіндік береді. Бұл есепте шексіз арифметикалық тізбекті шығаратын генератор құру керек (мысалы, натурал сандар).

Тапсырманың мақсаты:

Берілген саннан басталып, берілген қадаммен арифметикалық тізбектің элементтерін тудыруды жалғастыратын генераторлық функцияны жазыңыз.

Шешімі:

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

"""

Шексіз арифметикалық реттілік генераторы. start – тізбектің бастапқы мәні

step – реттілік қадам

""" current = start while True:

yield current

current += step

# # Генераторды пайдалану мысалы sequence = infinite\_arithmetic\_sequence(1, 1)

# Начальное число 1, шаг 1

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

print(next(sequence))

Түсініктеме:

"Шексіз\_арифметикалық\_тізбелік" функциясы "бастау" санынан басталатын және "келесі()" шақырылған сайын оны "қадамға" шексіз арттыратын генератор болып табылады. `yield` кілт сөзі реттіліктегі келесі мәнді қайтару және келесі қоңырауға дейін функцияны кідірту үшін пайдаланылады.

`келесі()`.

Бұл шешім Python-да жалқау бағалауды пайдалануды көрсетеді. Бүкіл ретті жадта жасаудың және сақтаудың орнына, генератор қажет болған жағдайда әрбір келесі мәнді есептейді. Бұл генераторлық тәсілді үлкен деректермен немесе ықтимал шексіз тізбектермен жұмыс істеу үшін әсіресе пайдалы етеді.