ҚАЗАҚСТАН РЕСПУБЛИКАСЫНЫҢ БІЛІМ ЖӘНЕ ҒЫЛЫМ МИНИСТРЛІГІ

СӘТБАЕВ УНИВЕРСИТЕТІ

ИАиИТ институты

Программная инженерия кафедрасы



ЛАБОРАТОРИЯЛЫҚ ЖҰМЫС

Тақырыбы: **«Функционалдық принциптерді қолдану арқылы модульдік код құру»**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Жұмысты орындау сапасы | Баға диапазоны | Орындалған  % |
| 1 | Орындалған жоқ | 0% |  |
| 2 | Орындалды | 0-50% |  |
| 3 | Материялдық өзіндік жүйелендіру | 0-10% |  |
| 4 | Талап етілген көлемде және көрсетілген мерзімде орындау | 0-5% |  |
| 5 | Қосымша ғылыми әдебиеттерді пайдалану | 0-5% |  |
| 6 | Орындаған тапсырманың ерекшелігі | 0-10% |  |
| 7 | СӨЖ-ді қорғау | 0-20% |  |
|  | Қорытынды: | 0-100% |  |

Оқытушы: Қасенхан А.М.

Студент: Нескенова А.К.

Мамандығы: Computer Science

Тобы: вт 12.10-14.05

Алматы 2024 ж

**7-зертханалық жұмыс:** Функционалдық принциптерді қолдану арқылы модульдік код құру

**Мақсат:** Python тілінде функционалдық бағдарламалау негізінде модульдік және кеңейтілетін кодты құру принциптері мен әдістерін меңгеру. Зертхананың мақсаты функционалдық ұғымдарды меңгеру ғана емес, сонымен қатар оларды таза, қолдауға болатын және масштабталатын кодты әзірлеу үшін қолдану болып табылады.

10. Оқиғаларды асинхронды өңдеу - Веб-қосымшада пайдаланушы оқиғаларын асинхронды өңдеу жүйесін құру

import asyncio

from aiohttp import web

#asyncio асинхронды бағдарламалау үшін пайдаланылады, aiohttp асинхронды HTTP бар веб-сервер жасау үшін пайдаланылады

async def handle(event\_processor):

async def inner\_handle(request):

event = await get\_next\_event()

await event\_processor(event)

return web.Response(text=f'Processed event: {event}')

return inner\_handle

#асинхронды функциясы, event\_processor - қабылданған оқиғаны өңдейтін функция

# ішкі асинхронды функция, request-клиенттен келетін сұрау нысаны

#келесі оқиғаны алу. аwait-орындау жалғастырмас бұрын аяқталуын күту

# оқиғаны өңдеу үшін event\_processor(event) функциясына беру

#оңделген оқиғаны көрсететін мәтіні бар веб-серверден HTTP жауапты қайтару

async def get\_next\_event():

await asyncio.sleep(1)

return "User action"

#asyncio.sleep(1) 1 секундқа кодты орындаудағы асинхронды үзілісті білдіреді, келесі оқиға алу алдында задержка

#1 секунд күткеннен кейін функция «Пайдаланушы әрекеті» жолын қайтарады

async def process\_event(event):

print(f"Processing event: {event}")

#event аргументін қабылдайтын process\_event асинхронды функциясы, қабылданған оқиғаны өңдейді

#оқиға туралы ақпаратты консольге басып шығарады, оқиғаның өңделіп жатқанын көрсететін хабарламаны көрсетеді

async def main():

async def event\_processor(event):

await process\_event(event)

#веб-серверді настройкалау және іске қосу үшін пайдаланылатын main() негізгі асинхронды функциясы

#асинхронды функциясы, веб-сервер жұмыс істеп тұрған кезде алынуы мүмкін оқиғаларды өңдеуге

#оқиғаның өңделіп жатқанын көрсететін хабарламаны басып шығаратын process\_event функциясына жіберіледі

app = web.Application()

app.router.add\_get('/', await handle(event\_processor))

#aiohttp кітапханасынан web.Application() конструкторын пайдаланып веб-бағдарлама объектін жасайды

#app.router қолданба маршрутизаторын білдіреді, ол маршрут өңдеушілерін қосу үшін

add\_get('/') түбірлік URL мекенжайына GET сұраулары үшін маршрут өңдеушісін қосады.

await handle(event\_processor) event\_processor аргументін қабылдайтын асинхронды өңдеу функциясын пайдалану

event\_processor оқиғаларды өңдейтін асинхронды функция

runner = web.AppRunner(app)

await runner.setup()

site = web.TCPSite(runner, 'localhost', 8080)

await site.start()

#Веб қолданбасын іске қосу үшін AppRunner экземплярын жасау, аргумент веб қолданба болып табылатын app

#AppRunner жұмыс істеуге дайындалуда

#Localhost және 8080 портындағы соединение тыңдайтын TCPSite экземплярын жасау, TCP сервер мен клиент (веб-браузер) арасында байланыс орнату және олардың арасында HTTP сұраулары мен жауаптарын тасымалдау

#веб-серверді іске қосу

print("Web server started")

await asyncio.sleep(3600)

#Веб-сервер жұмыс істеп тұрғанын көрсететін «Веб-сервер іске қосылды» хабары

#3600 секунд (1 сағат) күту, сервер іске қосылғаннан кейін бір сағаттан кейін тоқтатылады

await main()

#main() негізгі асинхронды функцияны іске қосу, маршруттарды орнату, веб-серверді жасау және іске қосу және веб-серверді өшірмес бұрын бір сағат күту (asyncio.sleep(3600) арқылы).



http://localhost:8080/

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Дайындық сұрақтары:**

**1тапс. Бағдарламалаудағы модульдік дегеніміз не және ол қандай артықшылықтар береді?**

Бағдарламалаудағы модульдік үлкен бағдарламалық жүйелерді кішірек, оңай ауыстырылатын модульдерге немесе компоненттерге бөлуді білдіреді. Бұл модульдер әдетте белгілі бір функцияларды орындайды немесе жүйенің белгілі аспектілерін өңдейді.

Модульдік тәсілдің артықшылықтары:

**Кодты қайта пайдалану:** Модульдік тәсіл кодты қайта пайдалануды жеңілдетеді, себебі функционалдылықты бағдарламаның әртүрлі бөліктерінде немесе тіпті әртүрлі жобаларда пайдалануға болатын бөлек модульдерге бөлуге болады.

**Жақсартылған кодты оқу мүмкіндігі:** Бағдарламаны модульдерге бөлу кодты түсінікті және оқылатын етеді. Бұл әзірлеушілерге бағдарламаның құрылымы мен функционалдығын тез түсінуге көмектеседі.

**Қатені оқшаулау:** егер бір модульде қате орын алса, бұл бүкіл бағдарлама жұмысын тоқтатады дегенді білдірмейді. Модульдік тәсіл қателерді белгілі бір модуль ішінде оқшаулауға мүмкіндік береді, бұл оларды табуды және түзетуді жеңілдетеді.

**Жеңілдетілген тестілеу:** Модульдерді бір-бірінен тәуелсіз тексеруге болады, бұл бағдарламаны тұтастай тестілеу процесін жеңілдетеді. Бұл қателерді тиімдірек және дер кезінде анықтауға және түзетуге мүмкіндік береді.

**2тапс. Python-да функция құрамын қалай жүзеге асырасыз және ол қандай есептер үшін қолайлы?**

Python тілінде функция құрамы стандартты функцияларды, лямбда функцияларын және үшінші тарап кітапханаларын пайдалануды қоса алғанда бірнеше жолмен жүзеге асырылуы мүмкін.

Python-дағы функционалдық композиция LEGO-мен құрылысқа ұқсайды. Әрбір функция бір LEGO кірпішіне ұқсайды. Үлкенірек және күрделірек нәрсе жасау үшін кірпіштерді біріктіргенде, бұл функция құрамы.

Бұл тәсіл күрделірек әрекетті орындау үшін белгілі бір тәртіпте орындалуы қажет бірнеше қарапайым қадамдар болған кезде пайдалы. Мысалы:

*Деректерді дайындау:* бірнеше деректерді түрлендіруді ретімен орындау үшін функция құрамын пайдалануға болады. Мысалы, тазалауды қажет ететін деректер болса, содан кейін сүзгіден өткізіп, соңында қажетті формаға түрлендіріңіз.

*Орындалу ағынын басқару:* Функция құрамы қандай қадамдарды қай ретпен орындау керектігін анықтау арқылы бағдарламаның орындалу ағынын басқаруға көмектеседі.

*Қатені өңдеу:* қателерді өңдеу тізбегін жасау үшін функция құрамын пайдалануға болады. Мысалы, қате жіберуі мүмкін функция болса, сол қатені өңдейтін басқа функцияны шақыру үшін функция құрамын пайдалануға болады.

*Қайта пайдалану:* функция құрамы кодты қайта пайдалануға көмектеседі. Сіз қарапайым функциялар жинағын жасай аласыз, содан кейін әртүрлі есептерді шешу үшін оларды әртүрлі комбинацияларда біріктіре аласыз.

Әрбір әдістің мысалдары:

**1. Стандартты функцияларды пайдалану:**

def add(x, y):

return x + y

def multiply(x, y):

return x \* y

def compose(f, g):

return lambda x, y: f(g(x, y), y)

composed\_function = compose(add, multiply)

result = composed\_function(2, 3) # Это эквивалентно add(multiply(2, 3), 3)

print(result) # Жауабы: 9

**2. lambda-функцияларын пайдалану:**

def compose(f, g):

return lambda x, y: f(g(x, y), y)

composed\_function = compose(lambda x, y: x + y, lambda x, y: x \* y)

result = composed\_function(2, 3) # Это эквивалентно (2 \* 3) + 3

print(result) # Жауабы: 9

**3. Funcy кітапханасын пайдалану (оны pip install funcy арқылы орнатамыз):**

from funcy import compose

def add(x, y):

return x + y

def multiply(x, y):

return x \* y

# Применяем частичное применение функции add

add\_partial = lambda x: lambda y: add(x, y)

composed\_function = compose(add\_partial(3), multiply)

result = composed\_function(2, 3) # Это эквивалентно add(multiply(2, 3), 3)

print(result) # Жауабы: 9

Funcy кітапханасы — функционалдық бағдарламалаудың ыңғайлы әдістерін қамтамасыз ететін Python тіліндегі функциялармен жұмыс істеуге арналған құралдар жинағы. Funcy ұсынған негізгі мүмкіндіктер мен функциялардың кейбірі мыналар:

Функция құрамы: compose, rcompose – тізбектегі бірнеше функцияларды біріктіруге және оларды ретімен қолдануға мүмкіндік береді.

Функцияларды ішінара пайдалану: partial, partial\_right - кейбір аргументтердің мәндерін бекіте отырып, барлардың негізінде жаңа функцияларды жасауға мүмкіндік береді.

**3тапс. Зертханада қандай асинхронды бағдарламалау әдістерін қолдандыңыз?**

Мен кодта қолданған әдістер мен функциялар:

async def: асинхронды функциялардың анықтамасы async def кілт сөзінен басталады. Бұл функцияның асинхронды жұмыс істеуіне, яғни бұғаттамайтын түрде орындауға мүмкіндік береді(в неблокирующем режиме).

await: await операторы басқа асинхронды әрекеттердің аяқталуын күту үшін асинхронды функциялар да пайдаланылады. Бағдарлама await операторына жеткенде, ол асинхронды операция аяқталғанша тоқтайды.

asyncio.sleep(): asyncio.sleep() функциясы асинхронды кодта кідіріс жасау үшін пайдаланылады. Бұл жағдайда ол get\_next\_event() функциясында оқиғаны алу кезіндегі кідірісті модельдеу үшін қолданылады.

web.Application: aiohttp.web модуліндегі бұл класстар веб-қосымшаны жасау және тиісінше HTTP жауаптарын жасау үшін пайдаланылады.

web.AppRunner() және web.TCPSite(): Бұл сыныптар aiohttp.web модулінен де TCP сайтын іске қосу үшін веб-бағдарламаны іске қосу және конфигурациялау үшін пайдаланылады.

Бұл кодта негізгі басқару циклі await main() шақыру арқылы орындалады.

**4тапс. Асинхронды бағдарламалау Python тіліндегі функционалдық стильмен қалай интеграцияланады?**

Асинхронды бағдарламалау Python тіліндегі функционалдық стильмен келесі жолдармен біріктірілуі мүмкін:

**Жоғары ретті асинхронды функцияларды пайдалану:** Функционалдық бағдарламалауда функциялар басқа функцияларды аргумент ретінде қабылдай алады немесе оларды нәтиже ретінде қайтарады. Бұл тәсіл асинхронды функциялар үшін де жұмыс істейді. Осылайша сіз басқа асинхронды функцияларды дәлел ретінде қабылдайтын немесе оларды нәтиже ретінде қайтаратын асинхронды функцияларды жасай аласыз.

**Асинхронды функциялардың функционалдық құрамы:** Функционалдық құрам - бұл бірнеше функцияларды бір функцияға біріктіру процесі. Бұл тәсілді асинхронды функцияларға да қолдануға болады. Асинхронды деректерді өңдеу үшін оларды бір-біріне ретімен қолдану арқылы асинхронды функцияларды тізбектей аласыз.

**Таза функциялар:** Функционалдық бағдарламалау стилі жанама әсерлері жоқ және тек олардың аргументтеріне тәуелді таза функцияларды құруға баса назар аударады. Бұл принципті асинхронды функцияларға да қолдануға болады, мұнда жанама әсерлерді азайту және сыртқы күйді өзгертуді болдырмау қажет болуы мүмкін.

**Функционалдық құрылымдарды пайдалану:** функционалдық бағдарламалаудың көптеген құралдары мен тұжырымдамалары асинхронды бағдарламалауда пайдалы болуы мүмкін. Мысалы, неғұрлым мәнерлі және таза асинхронды кодты жасау үшін функция құрамы, карриинг, монадтар және т.б.

**Функционалдық кітапханалар:** Python үшін асинхронды контексте пайдалануға болатын функционалдық бағдарламалау құралдары мен конструкцияларды қамтамасыз ететін кітапханалар бар. Мысалы, funcy, toolz, fn.py және басқа кітапханалар асинхронды кодпен жұмыс істеу кезінде пайдалы болуы мүмкін әртүрлі функциялар мен утилиталарды ұсынады.

Осылайша, асинхронды бағдарламалау Python тіліндегі функционалдық стильмен оңай біріктіріліп, асинхронды операцияларды өңдеу үшін мәнерлі, модульдік және таза код жасау мүмкіндігін береді.

**5тапс. Модульдік бағдарламалауда қандай мәселелер туындауы мүмкін және оларды шешу жолы?**

**Шамадан тыс күрделілік:** Кейбір модульдер шамадан тыс логикаға немесе функционалдылыққа байланысты тым күрделі болуы мүмкін, бұл оларды түсіну мен қолдауды қиындатады. Шешуі: Күрделі модульдерді әрқайсысы белгілі бір тапсырмаға жауап беретін қарапайым және кішірек модульдерге бөліңіз және жауапкершілікті бөлу принципін қолдану.

**Тым үлкен модульдер:** Модульде тым көп функционалдылық болса, бұл түсіну және қызмет көрсетуді қиындатады. Шешуі: Үлкен модульдерді кішірек және басқарылатын құрамдастарға бөлу, олардың әрқайсысы белгілі бір мәселені шешеді.

**Басқа модульдермен үйлеспеушілік:** Кейбір модульдер интерфейстердегі немесе пайдаланылған кітапхана нұсқаларындағы айырмашылықтарға байланысты үйлесімсіз болуы мүмкін. Шешім: Модульдер арасындағы өзара әрекеттесу үшін нақты ережелерді орнату және кітапхана нұсқаларын басқару.

**Тестілеудің қиындығы:** Кейбір модульдерді күрделі логикаға немесе тәуелділікке байланысты тексеру қиын болуы мүмкін. Шешім: Әр модульдің әртүрлі пайдалану жағдайларында дұрыс жұмыс істейтініне көз жеткізу үшін бірлік сынақтарын жазу.

**Құжатсыз код:** Модульде жеткілікті түсініктемелер мен құжаттама болмаса, бұл басқа әзірлеушілерге түсіну мен пайдалануды қиындатады. Шешім: Басқа әзірлеушілерге түсінуді жеңілдету үшін әрбір функцияға, классқа және кодтың маңызды бөлімдеріне түсініктемелер қосу арқылы кодты құжаттау.

**Ресурстарды тиімсіз пайдалану:** модуль жад, процессор уақыты немесе желілік қосылымдар сияқты ресурстарды тиімді пайдаланбаса, бұл қолданбаның нашар өнімділігі мен сенімсіздігіне әкелуі мүмкін. Шешім: Тиімді алгоритмдер мен деректер құрылымдарын пайдаланып кодты оңтайландыру.

**6тапс. Функция құрамының кодты қайта пайдалану мүмкіндігін арттырудағы рөлі қандай?**

Функция құрамының кодты қайта пайдалану мүмкіндігін арттырудағы рөлі кодты қайта жазусыз әртүрлі контексттерде пайдалануға болатын икемді және модульдік құрамдастарды жасау болып табылады. Функция құрамы кодты қайта пайдалану мүмкіндігін қалай жақсартады:

**Жақсартылған кодты оқу мүмкіндігі:** Функция құрамы кодты оқылатын және түсінікті ете алады. Көп нәрсені орындайтын бір ұзақ, күрделі функцияның орнына оны нақты тапсырмаларды орындайтын кішірек, басқарылатын функцияларға бөлуге болады. Бұл кодты құрылымды және түсінуді жеңілдетеді.

**Түзетуді және тестілеуді жеңілдетеді:** Функция құрамы кодты түзету және тестілеу процесін жеңілдетуге көмектеседі. Әрбір функция бөлек тапсырманы орындайтындықтан, олардың дұрыс жұмыс істеп тұрғанын оңай тексере аласыз, сондай-ақ әрбір функцияны бөлек тексереді.

**Икемділік пен кеңейтімділік арттырады:** Функция құрамын пайдалану кодыңызды икемді және оңай кеңейтетін етеді. Негізгі кодқа әсер етпестен композицияға жаңа функцияларды ауыстыру немесе қосу арқылы бағдарламаның әрекетін оңай өзгертуге болады. Бұл кодқа өзгертулер енгізуді жеңілдетеді және оны ауқымды етеді.

**Модульдік және абстракциялық:** Функция құрамы күрделі есептерді қарапайым және кішірек ішкі мәселелерге бөлуге мүмкіндік береді, олардың әрқайсысы жеке функция арқылы шешіледі. Бұл түсінуге және пайдалануға оңай модульдік кодты алға жылжытады.

Осылайша, функция композициясы жобаңызды әзірлеуді және қолдауды жеңілдететін модульдік, икемді және қайта пайдалануға болатын кодты жасауға көмектеседі.

**7тапс. Асинхронды бағдарламалау практикалық емес мысал келтіре аласыз ба?**

Асинхронды бағдарламалау практикалық болмауы мүмкін мысал:

Сізде минутына аздаған сұрауларға қызмет көрсететін веб-бағдарлама бар деп елестетіңіз. Әрбір сұрау ең аз өңдеу уақытын талап етеді және басқа сұрауларға тәуелсіз. Бұл жағдайда асинхронды бағдарламалауды пайдалану қажетсіз және практикалық болмауы мүмкін.

Неліктен бұлай:

**Жүйе жүктемесі төмен:** қолданба аздаған сұрауларға қызмет етеді және бір уақытта көп сұрауларды өңдеу үшін оңтайландырудың қажеті жоқ.

**Кодтың қарапайымдылығы:** синхронды кодты жазу, түсіну және қолдау оңайырақ болуы мүмкін, әсіресе асинхронды операциялар қажет болмаса.

**Бұғаттау әрекеттері жоқ:** сұраулар жылдам өңделеді және басқа сұрауларға немесе енгізу/шығаруға тәуелді емес, сондықтан блоктауды немесе күтуді басқару үшін асинхронды әрекеттерді пайдаланудың қажеті жоқ.

**Қарапайым және сызықтық тапсырмалар**: Егер бағдарлама параллельді орындауды немесе көптеген сыртқы сұрауларды күтуді қажет етпейтін қарапайым және сызықтық тапсырмаларды орындаса, асинхронды бағдарламалауды пайдалану артық болуы және қосымша пайдасыз кодқа күрделілік қосуы мүмкін.

**Асинхронды кітапханалардың болмауы:** Егер бағдарламалау тіліңізде тапсырмаларды асинхронды түрде орындауға арналған тиімді және қолдау көрсетілетін асинхронды кітапханалар болмаса, асинхронды бағдарламалау қиын және мүмкін емес болуы мүмкін.

Мұндай жағдайларда синхронды бағдарламалауды пайдалану оңайырақ және ыңғайлырақ болуы мүмкін. Бұл жағдайда асинхронды бағдарламалауды пайдалану нақты өнімділік немесе масштабтау артықшылықтарынсыз қажетсіз код күрделілігін қосуы мүмкін. Оның орнына синхронды бағдарламалауды пайдалану қарапайым және тиімдірек тәсіл болуы мүмкін.

**8тапс. Асинхронды қолданбаларда күйді қалай басқарасыз?**

Асинхронды қолданбалардағы күйді басқарудың мақсаты ағымдағы бағдарлама күйі туралы ақпаратты болжауға болатын, қауіпсіз және оңай басқарылатын жолмен сақтау және өзгерту болып табылады.

Асинхронды қолданбалардағы күйді басқарудың мақсаты - ағымдағы жағдай туралы ақпаратты қадағалауға және өңдеуге оңай жолмен сақтау. Міне, күйді басқарудың бірнеше қарапайым тәсілдері:

**Күй нысандарын(объекты) пайдалану:** Біз қолданбамызды күй оқиғалар немесе әрекеттер негізінде өзгерте алатын нысандар арқылы ұсынылатындай етіп ұйымдастыра аламыз. Мысалы, егер біз ойын жазып жатсақ, күйлер «ойын басталды», «ойын кідіртілді», «ойыншы жеңді» т.б. қамтуы мүмкін. Осы күй нысандарының әрқайсысында күйлер арасындағы оқиғалар мен ауысуларды өңдеу логикасы бар.

**Бақылаушы үлгісін пайдалану:** Бұл үлгі нысандарға басқа нысандардағы өзгерістер туралы ескертуге және оларға жауап беруге мүмкіндік береді. Қолданбамыздың әртүрлі бөліктеріне күй өзгерістеріне жазылуға және сәйкесінше жауап беруге рұқсат беру арқылы күйді басқару үшін осы үлгіні қолдана аламыз.

**Өзгермейтін деректерді пайдалану:** ақпарат айнымалыларда сақталған кезде, бұл айнымалыларды тікелей өзгерту мүмкін емес. Оның орнына, бірдеңені өзгерту қажет болса, біз қажетті өзгерістермен деректердің жаңа көшірмесін жасаймыз. Бұл күтпеген жанама әсерлерді болдырмауға көмектеседі.

**Деректер ағындарын пайдалану:** біз қолданбамызды деректердегі өзгерістерге немесе жүйеде болып жатқан оқиғаларға әрекет ететіндей ұйымдастыра аламыз. Мысалы, біз қолданбамыздың күйі туралы ақпаратты қамтитын деректер ағынын жасай аламыз және оларға жауап беру және пайдаланушы интерфейсін жаңарту немесе басқа әрекеттерді орындау үшін сол ағындағы өзгерістерге жазыла аламыз.

**9тапс. Функционалдық үлгілер дегеніміз не және олар сіздің зертханалық жұмысыңызда қалай қолданылды?**

**Функционалды үлгілер (немесе функционалдық шаблондар**) - негізгі құрылымдық блок ретінде функцияларды пайдалануға және мәселелерді шешу үшін біріктіруге негізделген бағдарламалау тәсілі. Олар функционалдық бағдарламалауда қолдануға болатын үлгілер немесе жалпы есептерді шешу әдістері. Бұл үлгілер сізге түсінікті, модульдік және тиімді код жазуға көмектеседі.

Зертханалық жұмыста келесі функционалдық үлгілер қолданылды:

**Замыкания (Closures):** handle функциясы inner\_handle айнымалысына сыртқы аумақтан қатынасатын кірістірілген inner\_handle функциясын жасайды және қайтарады. Бұл event\_processor функциясын орындалу контекстін сақтай отырып, handle беруге мүмкіндік береді.

**Жоғары ретті функциялар:** handle функциясы event\_processor аргументін қабылдайды, ол өзі функция болып табылады. Сонымен, handle басқа функциямен жұмыс істейтін жоғары ретті функция ретінде қарастыруға болады (event\_processor).

**Функция құрамы, композиция функций:** inner\_handle функциясы get\_next\_event, event\_processor функцияларын ретімен шақырады, содан кейін нәтижені қайтарады. Мұны осы функциялардың құрамы ретінде қарастыруға болады, мұнда бір функцияның шығысы екіншісіне кіріс ретінде пайдаланылады.

**Асинхронды функциялар және await:** Барлық код асинхронды функцияларды және await операторын пайдалануға негізделген, бұл бағдарламаға енгізу/шығару әрекеттерін (келесі оқиғаны қабылдау немесе оқиғаны өңдеу сияқты) аяқтау үшін асинхронды түрде күтуге мүмкіндік береді.

**Функционалды маршрутизатор әдістері:** код арнайы URL мекенжайына сұрау өңдеушісін қосу үшін add\_get маршрутизаторының функционалдық әдістерін пайдаланады.

**10тапс. Қандай Python құралдары мен кітапханалары асинхронды бағдарламалауға қолдау көрсетеді?**

Python асинхронды бағдарламалау үшін бірнеше құралдар мен кітапханаларды ұсынады.

**asyncio:** Бұл асинхронды кодты жазу үшін негізді қамтамасыз ететін стандартты Python кітапханасы. Ол сәйкесінше асинхронды функцияларды анықтау және күту үшін async және await кілт сөздерін қамтиды.

**aiohttp:** Бұл asyncio көмегімен желілік қолданбалар мен серверлерді құруға арналған кітапхана. Ол асинхронды веб-қосымшалар мен API интерфейстерін жазу үшін кеңінен қолданылады.

**aiofiles:** Бұл асинхронды код үшін асинхронды файл енгізу/шығару әрекеттерін қамтамасыз ететін кітапхана. Ол оқу, жазу және жою сияқты асинхронды файл операцияларын орындауға мүмкіндік береді.

**aiohttp- client:** Бұл асинхронды Python кодынан асинхронды HTTP сұрауларын жасауға мүмкіндік беретін asyncio арналған HTTP клиенті.

**trio:** Бұл пайдаланудың қарапайымдылығына және таза API интерфейсіне бағытталған асинхронды бағдарламалау кітапханасы. Ол асинхронды операциялар мен оқиғаларды ұйымдастыру механизмдерін қамтамасыз етеді.

Бұл Python тіліндегі асинхронды бағдарламалауға арналған кітапханалар мен құралдардың бірнеше мысалдары ғана. Олардың әрқайсысының өзіндік сипаттамалары бар және әртүрлі мақсаттарға арналған, сондықтан таңдау жобаның нақты талаптарына байланысты.

**11тапс. Модульдік тәсіл тестілеу және жөндеу бағдарламаларына қалай әсер етеді?**

**Тестілеуді жеңілдету:** Модульдік тәсілді пайдалану кезінде бағдарлама шағын модульдерге бөлінеді, олардың әрқайсысы белгілі бір функцияны орындайды. Бұл тестілеуді ыңғайлы етеді, өйткені модульдерді бір-бірінен тәуелсіз тексеруге болады. Бұл қателерді анықтауды және әрбір модульдің дұрыс жұмысын бөлек тексеруді жеңілдетеді.

**Жақсартылған сенімділік:** Модульдік тәсіл сенімдірек бағдарламаларды жасауға көмектеседі, себебі әрбір модульді оқшаулап тексеруге болады. Бұл қателердің ықтималдығын азайтады және оларды анықтау мен түзетуді жеңілдетеді.

**Оңай жөндеу:** Модульдік тәсілмен, егер бағдарлама дұрыс жұмыс істемесе, модульдерде нақты анықталған функционалдық аймақтар болғандықтан, ақауды тудыратын модульді анықтау оңайырақ. Бұл жөндеу процесін жеңілдетеді және қателерді табуға және түзетуге кететін уақытты азайтады.

**Өзгертудің қарапайымдылығы:** Әрбір модульде шектеулі жауапкершілік ауқымы және нақты анықталған кірістер мен шығыстар болғандықтан, бағдарламаға өзгертулер болжамды және қауіпсіз болады. Себебі бір модульдегі өзгерістер басқа модульдерде күтпеген жанама әсерлерді сирек тудырады.

**Әзірлеудің қысқартылған уақыты:** Модульдік тәсіл шағын модульдерден бастап, оларды үлкен жүйелерге дәйекті түрде біріктіре отырып, бағдарламаларды кезең-кезеңімен әзірлеуге мүмкіндік береді. Бұл әзірлеу тобының мүшелері арасында жұмысты бөлуге мүмкіндік береді және бағдарламаны тұтастай әзірлеуге қажетті уақытты қысқартады.

**12тапс. Функционалдық стильде асинхронияны жүзеге асыру кезінде қандай қиындықтарға тап болдыңыз?**

**Кодты түсіну қиындығы:** Кейде асинхронды кодтың қалай жұмыс істейтінін түсіну қиын болуы мүмкін, әсіресе оның ішінде көптеген функциялар мен қоңыраулар болған кезде. Бұл бағдарламаның қалай жұмыс істейтінін және қандай деректерді өңдейтінін түсінуді қиындатады.

**Уақытты басқару және күту мәселелері:** Асинхронды кодта серверден жауапты күту сияқты әрекеттердің аяқталуын жиі күтуге тура келеді. Уақыт пен күтуді басқару функционалдық стильде қиын болуы мүмкін, себебі функциялар әдетте таза және сыртқы факторларға тәуелді емес болуы керек.

**Функциялық композицияларды пайдалану қиындығы:** Функция композициялары функцияларды біріктіру үшін пайдалы болғанымен, асинхронды операцияларды және олардың нәтижелерін басқару қажеттілігіне байланысты асинхронды кодта күрделі болуы мүмкін.

**13тапс. Таза асинхронды кодты жазуға функционалдық бағдарламалау қалай ықпал ететінін түсіндіре аласыз ба?**

**Таза функциялар:** Функционалдық бағдарламалауда функциялар таза, яғни олардың жанама әсерлері жоқ және бірдей кіріс берілгенде әрқашан бірдей нәтижені қайтарады. Бұл функцияларды тексеруді жеңілдетеді, бұл кодты жөндеуді жеңілдетеді.

**Өзгермейтін деректер құрылымдарын пайдалану:** Функционалды бағдарламалау көбінесе өзгермейтін деректер құрылымдарын пайдаланады, олар деректер жасалғаннан кейін өзгертілмейтінін қамтамасыз етеді.

**Жоғары ретті функциялар және композиция:** Функционалды бағдарламалау барлардан жаңа функцияларды жасау үшін жоғары ретті функцияларды және функция құрамын пайдалануға мүмкіндік береді. Бұл асинхронды қолданбаларды жазуды және қолдауды жеңілдететін модульдік және қайта пайдалануға болатын кодты жазуға көмектеседі.

**Оңай параллелизация:** Функционалды тілдер map-reduce азайту операциялары және орындаудың параллель ағындары сияқты есептеулерді параллельдеу үшін ыңғайлы құралдарды ұсынады.

Барлық осы функционалдық бағдарламалау принциптері техникалық қызмет көрсетуге, тексеруге және жөндеуге оңай, таза, асинхронды код жазуды ынталандырады.

**Модульдік және функция құрамы:** Бұл асинхронды кодты оңай біріктіріліп, қайта пайдалануға болатын шағын, тәуелсіз функцияларға бөлуге мүмкіндік береді. Бұл тәсіл кодты жақсырақ ұйымдастыруға ықпал етеді және қателер ықтималдығын азайтады.

**Функционалдық үлгілерді пайдалану:** Функционалдық бағдарламалау каррирование, ішінара функцияны қолдану және рекурсия сияқты әртүрлі функционалдық үлгілерді пайдалануды ынталандырады. Бұл үлгілер асинхронды кодты жазуды жеңілдетіп, түсінікті және мәнерлі ете алады.

**14тапс. Асинхронды функцияларды құру кезінде қандай негізгі шақырулар болды?**

Кодта асинхронды функция құрамының негізгі шақыруы inner\_handle функциясының ішінде күтуге шақыру болып табылады:

**async def inner\_handle(request):**

**event = await get\_next\_event() # Вызов асинхронной функции get\_next\_event**

**await event\_processor(event) # Вызов асинхронной функции event\_processor**

**return web.Response(text=f'Processed event: {event}')**

Бұл шақырулардың әрқайсысы асинхронды болып табылады және асинхронды inner\_handle функциясының контекстінде орындалады. Бұл бірнеше асинхронды әрекеттерді бір функция ішінде біріктіруге және олардың ретімен орындалуын қамтамасыз етуге мүмкіндік береді.

Асинхронды функция құрамының тағы бір негізгі тұсы негізгі функциядағы веб-қосымшаға сұрау өңдеушісін қосқанда await функциясын шақыру болып табылады:

**app.router.add\_get('/', await handle(event\_processor))**

Мұнда await handle(event\_processor) — асинхронды inner\_handle функциясын қайтаратын асинхронды handle функциясына шақыру. Сондықтан асинхронды inner\_handle функциясы белгілі бір URL '/' үшін сұрау өңдеушісі ретінде пайдаланылады. Ол сондай-ақ асинхронды функциялардың құрамын білдіреді, мұнда бір функция белгілі бір функционалдылықты жүзеге асыру үшін басқасының бөлігі ретінде пайдаланылады.

**15тапс. Модульдік асинхронды қолданбаның архитектурасын қалай жақсартуға болады?**

**Кодты модульдеу:** қолданбаны функционалдылыққа немесе сұрау түріне қарай модульдерге бөлу оны модульдік және түсінуді жеңілдетеді. Әрбір модуль өзінің асинхронды функцияларын және класстарын қамтуы мүмкін.

**Кірістірілген жоғары ретті функцияларды пайдалану**: Python-да map(), filter() және reduce() сияқты көптеген кірістірілген жоғары ретті функциялар бар, бұл кодты ықшам әрі жеңілірек етеді.

**Генераторларды пайдалану:** Python-дағы генераторлар деректердің жалқау тізбектерін жасауға мүмкіндік береді. Олар код көлемін азайтуға және өнімділікті жақсартуға көмектеседі.

**Декораторларды пайдалану:** Декораторлар бастапқы кодын өзгертпестен функцияларға қосымша функционалдылықты қосуға мүмкіндік береді. Олар нәтижелерді кэштеу, тіркеу және басқа да көптеген тапсырмалар үшін пайдаланылуы мүмкін.

**Кірістірілген модульдер мен кітапханаларды пайдалану:** Python қосымша код жазусыз көптеген тапсырмаларды орындауға көмектесетін көптеген кірістірілген модульдер мен кітапханаларды ұсынады. Код өлшемін азайту үшін оларды пайдалана аламыз.