# 6-зертхана: Функционалды реактивті бағдарламалау (FRP)

**Мақсат:**

Python контекстінде функционалдық реактивті бағдарламалау (FRP) принциптерін үйреніңіз және қолданыңыз. Жұмыстың мақсаты - деректер ағындарын, реактивті айнымалыларды және асинхронды оқиғаларды өңдеуді қоса алғанда, FRP негізгі тұжырымдамаларын түсіну және жауап беретін және модульдік қосымшаларды жасау үшін осы тұжырымдамаларды пайдалану дағдыларын дамыту.

Тапсырмалар:

1. FRP негіздері:
   * Ағындар, сигналдар және реактивті айнымалылар сияқты негізгі FRP тұжырымдамаларын үйреніңіз.
   * реактивті өзгерістерді тарату және тәуелділікті басқару механизмдерін талдау.
2. Қарапайым FRP сценарийлерін іске асыру:
   * өзгермелі деректерді өңдеу және көрсетумен байланысты мәселелерді шешу үшін FRP қолдану.
   * Пайдаланушы әрекеттеріне немесе сыртқы оқиғаларға жауап ретінде реактивті деректерді жаңартуды көрсететін мысалдарды әзірлеу.
3. Қолданыстағы қолданбалармен интеграция:
   * Қолданыстағы Python қолданбаларына FRP тәсілдерін біріктіру жолдарын қарастыру.
   * FRP қолдану арқылы жауап беруді жақсарту және код күрделілігін азайту мүмкіндіктерін талдау.
4. FRP-дегі қосымша тақырыптар:
   * Қателерді өңдеу, ағынды біріктіру және асинхронды операцияларды қоса алғанда, FRP жүйесіндегі озық әдістер мен үлгілерді үйреніңіз.
   * көптеген деректер көздерін тиімді өңдеуге қабілетті күрделі реактивті жүйелерді жасау.
5. Сыни талдау және рефлексия:
   * Нақты әлемдегі қолданбалар контекстінде FRP артықшылықтары мен шектеулерін бағалау.
   * Нақты бағдарламалық тапсырмалар үшін тәсілдер мен құралдарды таңдауда сыни тұрғыдан ойлауды дамыту.

Зертханалық жұмыстың маңыздылығы:

Зертхана студенттерді қазіргі заманғы бағдарламалық қамтамасыз етуді әзірлеуде маңыздырақ болып келе жатқан функционалдық реактивті бағдарламалау парадигмасымен таныстыруға арналған. FRP интерактивті және асинхронды қолданбаларды құруға арналған талғампаз шешімдерді ұсынады, бұл күй мен деректер ағынын басқаруды жеңілдетеді. Жұмыс реактивті жүйелерді терең түсінуді дамытады және анағұрлым жауап беретін және модульдік қосымшаларды жобалау дағдыларын жетілдіреді.

# Жеке тапсырмалар:

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

Функционалды реактивті бағдарламалау (FRP) өзгерістерге жауап беретін жүйелерді құру үшін реактивті және функционалды бағдарламалау идеяларын біріктіреді.

1. FRP істер тізімі
   * Тапсырмаларды қосуға және жоюға жауап беретін FRP көмегімен қарапайым істер тізімі қолданбасын жасаңыз.

import tkinter as tk  
from rx import Observable  
from rx.subject import Subject  
  
class TodoList:  
 def \_\_init\_\_(self):  
 self.tasks = []  
 self.task\_added = Subject()  
 self.task\_removed = Subject()  
  
 def add\_task(self, task):  
 self.tasks.append(task)  
 self.task\_added.on\_next(task)  
  
 def remove\_task(self, task):  
 if task in self.tasks:  
 self.tasks.remove(task)  
 self.task\_removed.on\_next(task)  
  
def main():  
 todo\_list = TodoList()  
  
 def print\_tasks():  
 tasks\_label.config(text="Тізім: " + ", ".join(todo\_list.tasks))  
  
 def add\_task():  
 task = entry.get()  
 if task:  
 todo\_list.add\_task(task)  
 entry.delete(0, tk.END)  
  
 def remove\_task():  
 task = entry.get()  
 if task:  
 todo\_list.remove\_task(task)  
 entry.delete(0, tk.END)  
  
 root = tk.Tk()  
 root.title("Todo List")  
  
 entry = tk.Entry(root)  
 entry.pack()  
  
 add\_button = tk.Button(root, text="Қосу", command=add\_task)  
 add\_button.pack()  
  
 remove\_button = tk.Button(root, text="Өшіру", command=remove\_task)  
 remove\_button.pack()  
  
 tasks\_label = tk.Label(root, text="Істер тізімі:")  
 tasks\_label.pack()  
  
 todo\_list.task\_added.subscribe(lambda task: print\_tasks())  
 todo\_list.task\_removed.subscribe(lambda task: print\_tasks())  
  
 root.mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

# 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. Функционалды реактивті бағдарламалау дегеніміз не және оны қандай жағдайларда қолдану керек?

- Мақсаты: FRP анықтамасын және оны қолдану сценарийлерін түсіну.

1. FRP негізгі ұғымдары қандай?
   * Мақсаты: Деректер ағындары және реактивті күй сияқты FRP құрылыс блоктары туралы білімді бағалау.
2. FRP оқиғаны өңдеудің дәстүрлі тәсілінен несімен ерекшеленеді?
   * Мақсат: FRP және басқа оқиғаларды өңдеу үлгілерінің арасындағы айырмашылықтарды түсіну.
3. FRP бағдарламалық жасақтаманы әзірлеушілерге қандай артықшылықтар береді?
   * Мақсаты: Күйді басқаруды жеңілдету және код модульділігі сияқты артықшылықтарды түсінуді тексеру.
4. FRP енгізу үшін қандай Python кітапханаларын пайдаландыңыз және неліктен?
   * Мақсаты: Оқушылардың қол жетімді құралдар туралы білімін және мәселені шешу үшін таңдауын бағалау.
5. FRP-де деректер ағынын жүзеге асыру жолын түсіндіре аласыз ба?
   * Мақсаты: Студенттің FRP-де деректер ағындарын құру және басқару механизмін түсінуін тексеру.
6. FRP асинхронды операцияларды қалай өңдейді?

* Мақсаты: Оқушылар FRP асинхронды кодты өңдеуге және жанама әсерлерді өңдеуге қалай мүмкіндік беретінін түсінеді.

1. FRP пайдалануда қандай қиындықтар болуы мүмкін және оларды қалай жеңуге болады?

Мақсаты: Оқушылардың мүмкін болатын қиындықтар мен оларды шешу жолдары туралы хабардарлығын анықтау.

1. Қолданбаңызда немесе ойыныңызда реактивті әрекетті қалай тексердіңіз?

- Мақсаты: Реактивті жүйелерді сынау әдістерін түсіну.

1. FRP-де өнімділік мәселелері қандай болуы мүмкін және оларды қалай оңтайландыруға болады?

- Мақсаты: Реактивті қолданбалардың өнімділігі және оңтайландыру әдістері туралы білімдерін бағалау.

1. FRP-де жалқау бағалау қолданылатын мысал келтіре аласыз ба?

- Мақсаты: Жалқау бағалауды түсінуді және оны FRP контекстінде қолдануды тексеру.

1. FRP қолданбадағы күйді басқаруға қалай әсер етеді?
   * Мақсаты: FRP жағдайды басқаруға әсерін және оның нақты уақыттағы өзгерістерін түсіну.
2. FRP-мен жұмыс істегенде қандай дизайн үлгілері пайдалы болуы мүмкін?
   * Мақсаты: FRP-ны тиімді толықтыратын дизайн үлгілері туралы білімді бағалау.
3. Қолданыстағы қолданбаларға FRP қалай біріктіруге болады?

- Мақсат: кодты толығымен қайта жазбай-ақ, ағымдағы жобаларға FRP біріктіру тәсілдерін түсіну.

1. FRP қолданбаңыздағы деректер ағындарына жазылудан бас тартуды қалай басқарасыз?

- Мақсаты: Реактивті қолданбаларда жазылудың өмірлік циклін дұрыс басқару туралы білімдерін тексеру.

1. Что такое Функциональное Реактивное Программирование (FRP) и в каких случаях его стоит использовать?

FRP - это метод программирования, который сосредотачивается на работе с потоками данных и событиями в реальном времени. Он использует функциональные подходы для работы с изменениями данных и событий. FRP особенно полезен в задачах, где важно реагировать на изменения данных или событий, например, в разработке веб-приложений с динамическими интерфейсами, в игровой разработке, в потоковой обработке данных и в других приложениях, требующих реактивности.

2. Каковы ключевые концепции FRP?

Основные концепции FRP включают в себя:

- Потоки данных: Потоки данных представляют собой непрерывные потоки информации, которые могут изменяться со временем.

- Реактивное состояние: Это состояние данных, которое автоматически обновляется в ответ на изменения в потоках данных.

- Обработка событий: FRP позволяет реагировать на события, такие как клики мыши, нажатия клавиш и другие пользовательские действия.

3. Чем FRP отличается от традиционного подхода к обработке событий?

В традиционном подходе к обработке событий программисты часто используют обратные вызовы или паттерн "наблюдатель" для реагирования на события. В FRP данные и события рассматриваются как непрерывные потоки, а не отдельные события. FRP позволяет описывать взаимосвязи между данными и автоматически обновлять состояние при изменении данных или событий.

4. Какие преимущества предоставляет FRP разработчикам программного обеспечения?

Преимущества FRP включают в себя:

- Более простое управление состоянием приложения.

- Более чистый и понятный код благодаря декларативному подходу.

- Улучшенная отзывчивость и адаптивность приложения к изменяющимся условиям.

- Более легкая обработка асинхронных операций и событий.

- Высокая степень композициональности, что упрощает переиспользование и расширение кода.

5. Какие библиотеки Python вы использовали для реализации FRP и почему?

В Python мы можем использовать библиотеки, такие как RxPY или PyFunctional, потому что они предоставляют удобные инструменты для работы с потоками данных и событиями. RxPY особенно полезен для создания реактивных потоков данных и управления ими.

6. Можете ли вы объяснить, как реализовать поток данных в FRP?

Да, в FRP мы можем создавать потоки данных с помощью функций и операторов, которые преобразуют и комбинируют потоки данных. Например, мы можем использовать оператор `from\_iterable` для создания потока данных из итерируемого объекта и операторы `map`, `filter`, `scan` для преобразования и обработки данных в потоке.

7. Как в FRP обрабатываются асинхронные операции?

В FRP асинхронные операции могут быть обработаны с помощью операторов, которые поддерживают асинхронное выполнение, таких как `flatMap`, `concatMap`, `mergeMap`. Эти операторы позволяют выполнять асинхронные операции и объединять результаты в потоке данных.

8. Каковы могут быть трудности при использовании FRP и как их преодолеть?

Некоторые трудности при использовании FRP могут включать сложность в понимании концепций FRP, управлении состоянием приложения и управлении асинхронными операциями. Эти трудности могут быть преодолены с помощью практики, изучения документации и применения лучших практик программирования.

9. Как вы тестировали реактивное поведение в вашем приложении или игре?

Реактивное поведение в FRP-приложении может быть протестировано с помощью юнит-тестов, интеграционных тестов и тестового окружения, которое специально разработано для тестирования реактивных систем.

10. Каковы могут быть проблемы с производительностью в FRP и как их оптимизировать?

Проблемы с производительностью в FRP могут возникать из-за большого количества потоков данных или сложных операций. Для их оптимизации можно использовать техники, такие как ленивые вычисления, кэширование результатов и асинхронные операции.

11. Можете ли вы привести пример, где ленивые вычисления используются в FRP?

В FRP ленивые вычисления могут использоваться, например, когда мы создаем поток данных, который не будет вычисляться до тех пор, пока не будет запрошен результат его выполнения.

12. Как FRP влияет на управление состоянием в приложении?

FRP делает управление состоянием более простым и ясным, позволяя нам создавать реактивные потоки данных, которые автоматически обновляются при изменении состояния.

13. Какие паттерны проектирования могут быть полезны при работе с FRP?

Некоторые паттерны проектирования, такие как Observer, Pub-Sub и MVVM, могут быть полезны при работе с FRP, помогая организовать код и управлять взаимодействием между компонентами.

14. Как FRP может быть интегрировано в существующие приложения?

FRP может быть интегрировано постепенно, начиная с маленьких частей приложения и постепенно расширяя его область применения. Мы можем использовать библиотеки FRP для реализации новых функций или модулей в существующем коде.

15. Как вы управляете отпиской от потоков данных в вашем FRP-приложении?

Мы можем управлять отпиской от потоков данных, используя методы отписки, предоставляемые библиотеками FRP, или же следя за жизненным циклом компонентов приложения и отписываясь от потоков данных при необходимости.

dispose()

# Пример для решения

Задача: Реактивное Отслеживание и Отображение Данных Температуры

Представьте, что у вас есть поток данных о температуре, поступающих из различных источников (например, датчиков температуры). Задача состоит в том, чтобы реализовать простую реактивную систему на Python, которая отслеживает эти данные в реальном времени и обновляет среднюю температуру при каждом новом поступлении данных.

Цель задачи:

Создать реактивную систему, которая подписывается на поток данных о температуре, вычисляет и отображает обновленное среднее значение температуры при каждом изменении в потоке.

Решение:

Для этой задачи мы можем использовать библиотеку RxPy (Reactive Extensions for Python), которая предоставляет инструменты для реактивного программирования.

Сначала установите RxPy, если она еще не установлена: pip install rx

Теперь давайте создадим решение:

import rx

from rx import operators as ops

# Эмуляция потока данных о температуре temperature\_data = rx.of(20, 21, 22, 21, 20, 19, 20, 21, 22, 23)

# Реактивное вычисление средней температуры average\_temperature = temperature\_data.pipe(

ops.scan(lambda acc, temp: (acc[0] + temp, acc[1] + 1), (0, 0)), ops.map(lambda acc: acc[0] / acc[1])

)

# Подписка и вывод результата average\_temperature.subscribe(

lambda avg\_temp: print(f"Средняя температура: {avg\_temp:.2f}°C")

)

#### Объяснение:

* Мы создаём поток `temperature\_data`, который эмулирует поступление данных о температуре.
* `pipe` используется для создания цепочки операторов, обрабатывающих поток данных.
* С помощью `ops.scan` мы накапливаем сумму температур и количество измерений для дальнейшего вычисления среднего значения.
* `ops.map` преобразует накопленные данные в среднее значение температуры.
* `subscribe` используется для отслеживания результатов и вывода обновленного среднего значения температуры.

Это решение демонстрирует основные принципы FRP: реактивное отслеживание изменений в данных и декларативное определение логики обработки данных. Оно показывает, как FRP позволяет легко реагировать на изменяющиеся данные и создавать отзывчивые приложения.